ANALIZA I VIZUALIZACIJA ZADATAKA IZ FIZIKE U PYTHONU

Dragičević, Kristijan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ODJEL ZA FIZIKU

KRISTIJAN DRAGIČEVIĆ

ANALIZA I VIZUALIZACIJA ZADATAKA IZ FIZIKE U PYTHONU

Završni rad

Osijek, 2019.

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ODJEL ZA FIZIKU

KRISTIJAN DRAGIČEVIĆ

ANALIZA I VIZUALIZACIJA ZADATAKA IZ FIZIKE U PYTHONU

Završni rad

Predložen Odjelu za fiziku Sveučilišta Josipa Jurja Strossmayera u Osijeku

radi stjecanja zvanja prvostupnika fizike

Osijek, 2019.

„Ovaj završni rad izrađen je u Osijeku pod vodstvom mentora doc. dr. sc. Darija Hrupeca u sklopu Sveučilišnog preddiplomskog studija fizike na Odjelu za fiziku Sveučilišta Josipa Jurja Strossmayera u Osijeku.“

SADRŽAJ

Sveučilište Josipa Jurja Strossmayera u Osijeku Završni rad

Odjel za fiziku

ANALIZA I VIZUALIZACIJA ZADATAKA IZ FIZIKE U PYTHONU

KRISTIJAN DRAGIČEVIĆ

Sažetak

U procesu rješavanja problemskih zadataka iz fizike, nastoji se pojednostaviti jedan problem kao cjelinu na više manjih dijelova koji zahtijevaju manje resursa za pronalazak pojedinačnih rješenja kako bi bio dostupan pregled u prirodne pojave koje se nalaze u pozadini problema. U procesu pojednostavljenja problema veliku ulogu ima vizualizacija promatranog sustava, jer vrlo lako može ukazati na poveznice između pojedinih dijelova koje možda na prvi pogled nisu primjetne. Najjednostavniji mogući primjer vizualizacije problema je bilješka ili skica olovkom na papiru, no takav pristup nije nužno zadovoljavajući ako se radi o složenijem sustavu, ili ako se u obzir uzimaju velike količine numeričkih ili drugih podataka. U tom slučaju koriste se pomagala poput računala, što opet za sobom povlači zahtjeve da postoji način zapisa podataka na računalu, nakon čega slijedi obrada istih, te na kraju prikaz povratnih informacija u obliku prepoznatljivom korisniku, pretežno u grafičkom obliku. Primjer primjene računala kao takvog pomagala je korištenje programskog jezika Python koji podržava različite strukture podataka, te za kojeg postoje dodaci koji omogućuju jasan grafički prikaz obrađenih podataka u željenom obliku.

(31 stranica, 10 slika, 1 tablica, 12 dodataka)

Rad je pohranjen u knjižnici Odjela za fiziku

Ključne riječi : *MatPlotLib / NumPy* / programiranje / Python / *ScyPy* / vizualizacija

Mentor : doc. dr. sc. Dario Hrupec

Ocjenjivač : doc. dr. sc. Dario Hrupec

Rad prihvaćen :

University Josip Juraj Strossmayer Osijek Bachelor of Physics Thesis

Department of Physics

ANALYSIS AND VISUALISATION OF PROBLEMS IN PHYSICS WITH PYTHON

KRISTIJAN DRAGIČEVIĆ

Abstract

In the process of solving problems in physics, there is a tendency to simplify or to dissolve the problem as a whole into multiple smaller parts which require less resources to solve individually, and in the process shine a light on the natural phenomena in the background. In the process of simplifying problems, visualisation is of big importance because it can point to connections between certain smaller parts which may not be obvious at first. The simplest example of visualisation is taking notes or making sketches on paper, but such an approach may not necessarily be enough in case the problem is complicated or if the amount of dana taken into consideration is large. In that case we use apparatus such as computers, but that also has requirements of its own, such as availability of a form of recording data on a computer, followed by processing the data, and finally displaying the results, preferably in a graphical format. An example of such apparatus is the programming language Python, which supports different types of data structures, and also has additions which allow for a clear display of processed data in a desirable form.

(31 pages, 10 figures, 1 table, 12 appendix)

Thesis deposited in Department of Physics library

Keywords : *MatPlotLib* / *NumPy* / programming / Python / *ScyPy* / visualisation

Supervisor : Dario Hrupec, Ph.D., Assistant professor

Reviewer : Dario Hrupec, Ph.D., Assistant professor

Thesis accepted :

Uvod

Za učinkovito rješavanje zadataka iz fizike, naročito onih težih, dobrodošla je intuicija za fiziku. Razvoj intuicije može se pospješiti kroz razmišljanje i raspravu o problemima iz fizike, odnosno kroz praksu i težnju napretku. Barem se takav zaključak nameće promotri li se rad brojnih generacija velikih mislilaca i znanstvenika u prošlosti. Naravno, prilično se neupitno današnje razdoblje može okarakterizirati kao doba ubrzanog razvoja i napretka u smislu tehnologija i širokog raspona primjene stečenih znanja. Stoga nije iznenađenje da se dostupni alati i pomagala sve više koriste kako bi se što brže razvila vještina postavljanja i rješavanja problemskih zadataka, s ciljem produbljenja razumijevanja prirodnih pojava koje leže u njihovoj pozadini.

Jedan od tih alata je programski kod kojim problem iz područja fizike možemo zapisati u računalu prepoznatljivom obliku, a zatim koristeći programske naredbe i metode analizirati i vizualizirati problem pomoću računala, u obliku koji možda ne bi bio moguć uz samo razmišljanje i raspravu. U tom smislu, primjena dostupnih alata i tehnologija vrlo lako može pružiti novi kut pogleda na probleme iz fizike, što je u nekim situacijama upravo potrebno za pronalazak rješenja.

Kod se može pisati u različitim programskim jezicima, primjerice u Pythonu koji u zadnje vrijeme postaje sve popularniji, ne samo u zajednici fizičara i astronoma širom svijeta, već se sve više primjenjuje i u procesu obrazovanja na gotovo svim razinama, bilo u vlastitom ili formalnom obrazovanju. U nastavku rada razmotren je, i pomoću nekoliko pokaznih primjera detaljnije opisan pristup analizi i vizualizaciji problema iz fizike primjenom upravo programskog jezika Python.

Programski jezik Python

Razvoj Pythona kao programskog jezika počeo je još krajem 80.-ih godina 20. stoljeća, što je relativno daleko u prošlosti s obzirom na prosječnu brzinu razvoja novih tehnologija u zadnjih nekoliko desetljeća. Od 1989. godine, kada je počela njegova aktivna implementacija uz operativni sustav *Amoeba*, Python je doživio velik broj proširenja i dodataka. Neke ključne točke su izdanje Python 0.9.0 u veljači 1991., slavna verzija Python 1.0 u siječnju 1994., zatim Python 2.0 u listopadu 2000., i konačno, u prosincu 2008. objavljena je verzija Python 3.0. Naravno, ne treba zaboraviti da je između svake od tih ključnih točaka postojao velik broj malenih „stepenica" u razvoju koje su periodički proširivale i povećavale mogućnosti i spektar primjena Pythona. Treba napomenuti da je u trenutku pisanja rada posljednja službena verzija Python 3.7.4, objavljena 8. srpnja 2019.

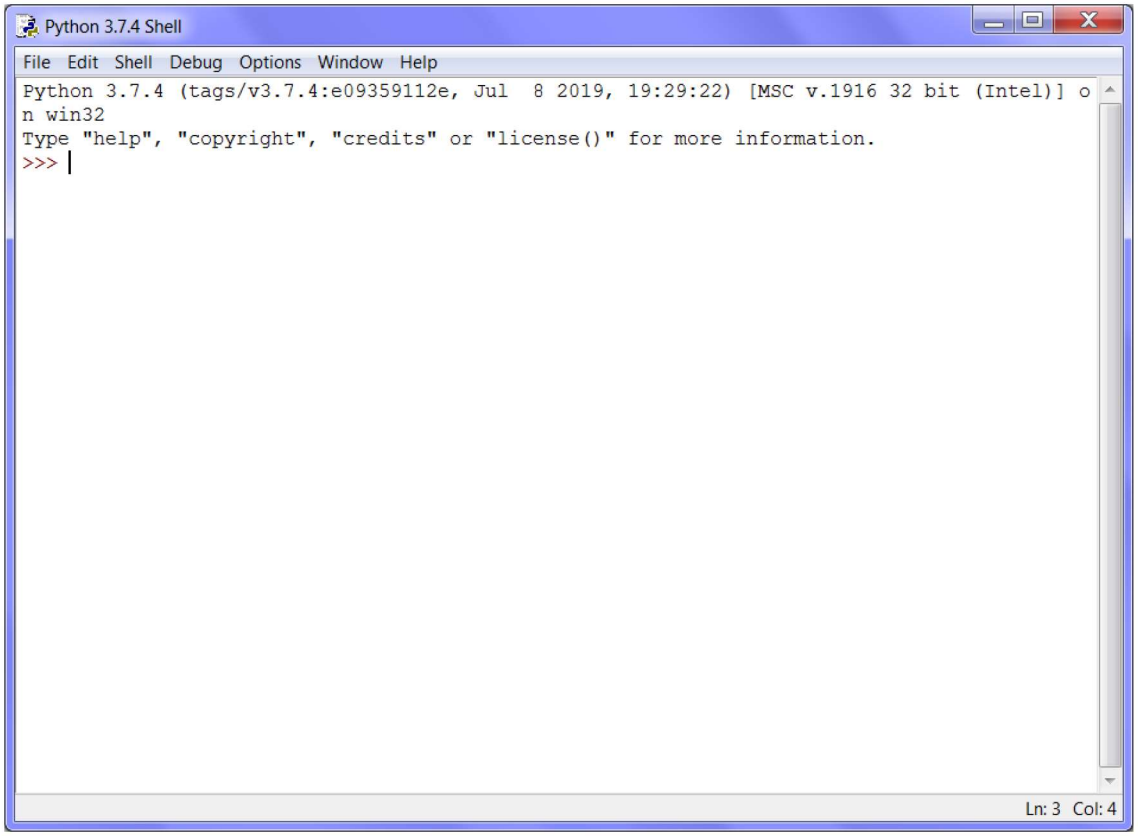
Slika 1: Python logotip

Uzimajući u obzir koliko dugo postoji, i koliko dugo se radi na njegovu razvoju, može se pretpostaviti da nije jednostavan zadatak pokušati ga samostalno ukratko opisati. Iz tog razloga postoje mnogi priručnici i vodiči koji su osmišljeni upravo s ciljem upoznavanja korisnika s osnovama funkcioniranja Pythona, ili čak u nekim slučajevima objašnjavanje složenijih struktura koje je moguće izraditi koristeći Python. Jedan takav priručnik je *„Introduction to Python for Science"* autora Davida Pinea. U uvodu priručnika, autor navodi kako je Python koristan za različite vrste znanstvenih problema, poput analiziranja i iscrtavanja podataka ili numeričkog rješavanja problema koje smatramo iznimno teškima ili čak nemogućima za riješiti analitičkim metodama. Također, treba napomenuti da to nisu jedine vrste upotrebe Pythona, jer se aktivno i učinkovito koristi u razne druge svrhe poput izrade mrežnih aplikacija ili obrade financijskih podataka, i slično.

Python je izrađen kao interpreter, što je jedna ključna karakteristika pri kategorizaciji programskih jezika. To znači da se u Pythonu može pokrenuti i izvoditi napisani programski kod bez da se prvo kompajlira, odnosno bez prevođenja programskog koda u strojni jezik. Upravo ta karakteristika dijeli programske jezike na interpretere i kompajlere. Također, bitno je naglasiti da je Python dinamički programski jezik, što znači da se pri pisanju koda ne mora ručno deklarirati varijable i definirati i izdvajati dijelove postojeće memorije računala prije pokretanja napisanog koda. Na kraju, možda i najbitnija stavka u opisu programskog jezika jest činjenica da je Python besplatan i svima dostupan na mrežnim stranicama, te su izrađene inačice programskog jezika i interpretera kompatibilne s većinom poznatijih računalnih operativnih sustava, uključujući, naravno, Microsoft Windows, MacOS i Linux. Upravo ta, možda malena ali definitivno ključna karakteristika čini Python jednim od poželjnijih programskih jezika među učenicima koji se tek upoznaju s programiranjem, ili točnije rečeno, profesori vrlo često naginju takvom tipu jezika pri odabiru i izradi nastavnog plana jer odabir jednostavnijeg i dostupnijeg programskog jezika definitivno ima utjecaja na razinu usvojenosti znanja i vještina kod učenika. Također, može se reći da postoji vrlo velika sličnost između sintakse naredbi u Pythonu i nekih od međunarodno poznatih jezika, u ovom slučaju engleskog jezika, što ga dodatno čini privlačnim učenicima koji, može se pretpostaviti, svakako upoznaju barem osnove engleskog jezika čak i prije nego dođu u dodir s programiranjem, bilo u sklopu školovanja ili vlastitim radom.

Korisničko sučelje i prilagođavanje

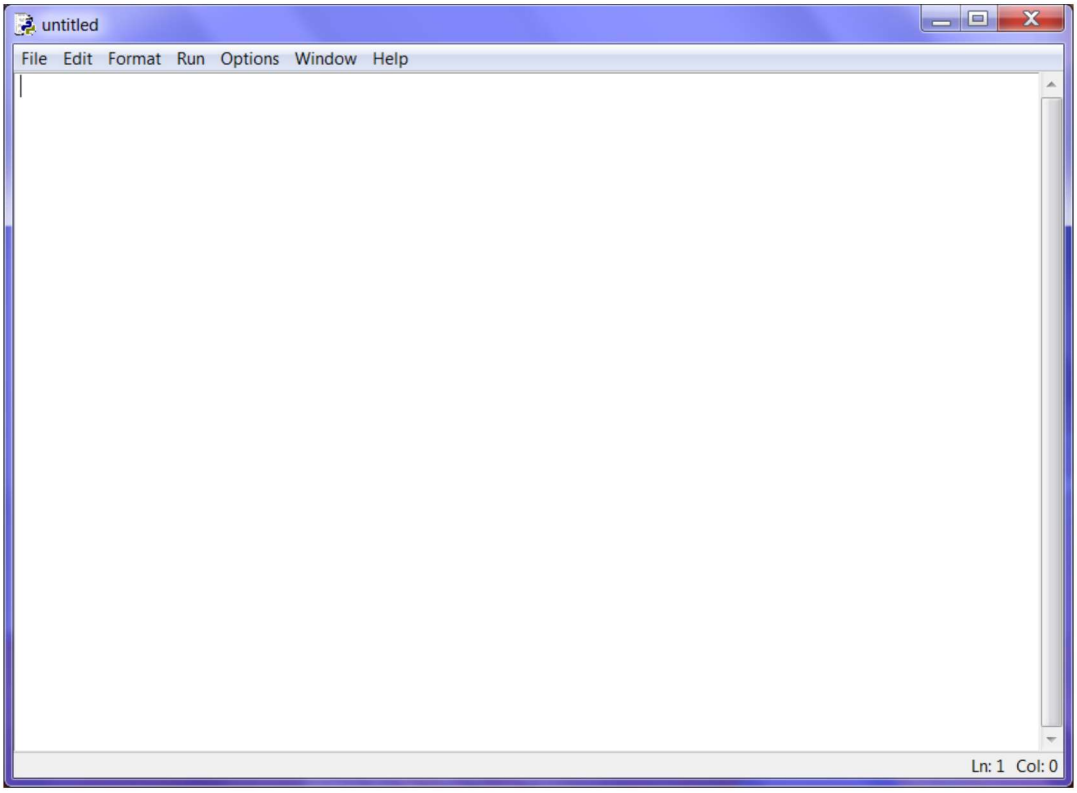
U radu nije dodatno objašnjen postupak instalacije i pokretanja samog programa Python, ali je u nastavku prikazan izgled korisničkog sučelja nakon uspješno odrađene instalacije i pokretanja aplikacije u operativnom sustavu Microsoft Windows 7. Nakon pokretanja otvara se početni prozor (eng. *Python Shell*) u kojem se može pisati kod, odnosno u kojem će biti ispisani i prikazani rezultati izvođenja napisanog koda.



Slika 2: Osnovni početni prozor u Pythonu (eng. Python Shell)

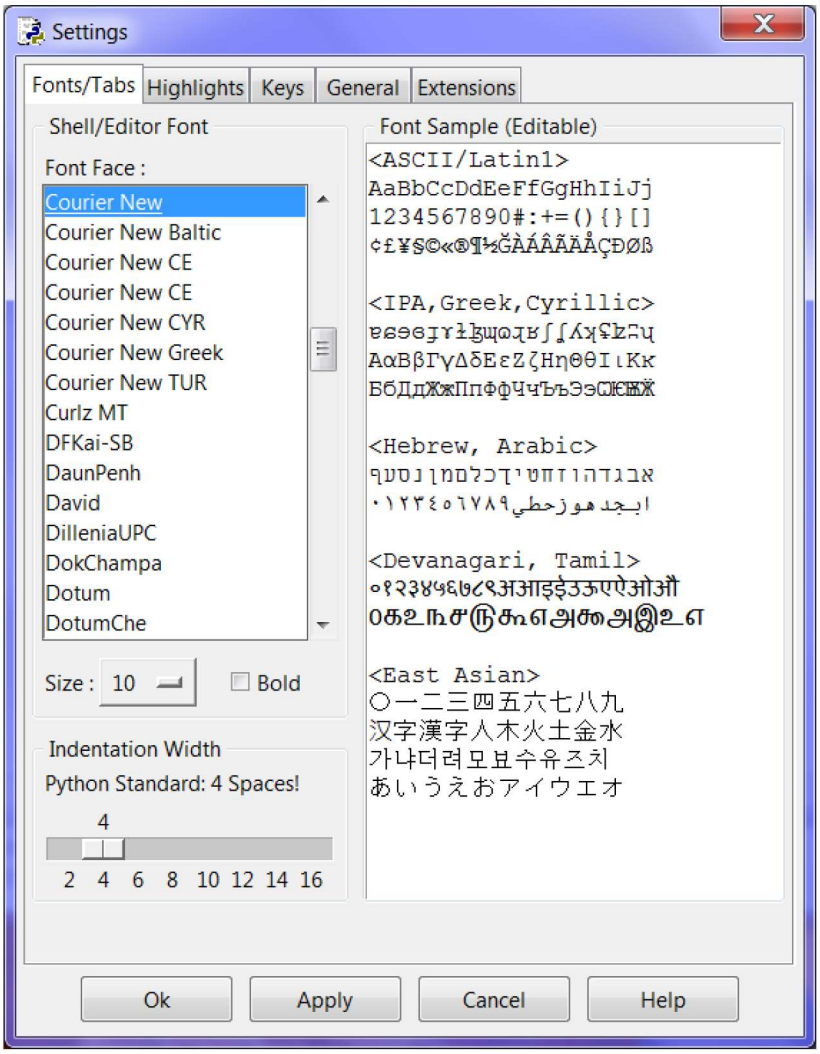
Treba naglasiti da nije preporučeno pisati kod u početnom prozoru jer početni prozor funkcionira na principu izvođenja koda nakon pritiska tipke *Enter* na tipkovnici, što znači da nije moguće napisati više od jednog reda koda, a tek nakon toga pokrenuti izvođenje. Iz tog razloga postoji mogućnost otvaranja novog prozora koji služi za pisanje većih elemenata koda i uređivanje istih u tekstualnom obliku. Po završetku pisanja koda u novom prozoru, možemo izvršiti napisani kod, te u početnom prozoru pregledati dobivene rezultate izvođenja koda, odnosno, ako postoji problem

s napisanim kodom koji onemogućuje uspješno izvođenje koda, bit će jasno naznačeni dijelovi koda koji stvaraju problem i bit će navedeno koje linije koda je potrebno prepraviti kako bi se kod uspješno pokrenuo.



Slika 3: Izgled novog prozora za pisanje koda

Nadalje, postoji mogućnost detaljnijeg prilagođavanja izgleda korisničkog sučelja Pythona prema vlastitim željama otvaranjem padajućeg izbornika *Options* i pritiskom na gumb *Configure IDLE*. Potom se otvara novi prozor s nekoliko kartica u kojima možemo urediti razne mogućnosti programa Python, od određivanja veličine početnog i novih prozora u pikselima, prilagođavanja fonta i boje teksta, definiranja veličine uvlaka za naredbe i petlje, do odabira boja kojima su naglašeni određeni segmenti koda poput pozvanih postojećih naredbi i slično.



Slika 4: Prozor mogućnosti (eng. Settings)

Dodatne biblioteke – moduli

Standardna inačica Pythona koja se dobije pokretanjem instalacije dostupne na mrežnim stranicama u sebi već sadrži određeni broj unaprijed definiranih dodatnih biblioteka naredbi koje su poznatije pod nazivom moduli. Moduli u Pythonu su skupovi unaprijed definiranih funkcija koje su opće korisne za pisanje koda u gotovo bilo koju svrhu, izrađeni s ciljem olakšavanja procesa pisanja koda krajnjim korisnicima. Treba naglasiti da su određeni moduli povezani s osnovnom inačicom Pythona, ali to ne znači da ih se može aktivno koristiti, točnije rečeno ne može se pozivati funkcije iz njih, bez da se eksplicitno pozovu ciljani moduli. Kako bi se pozvao bilo koji modul mora se poznavati točan naziv modula, odnosno ako se ne želi učitati cijeli modul, može se pozvati samo određene funkcije iz odabranog modula, za što opet treba poznavati točan naziv funkcije i modula kojem funkcija pripada. U donjoj tablici prikazani su nazivi nekih od češće korištenih ili općenito poznatijih modula dostupnih u standardnoj inačici Pythona, zajedno s kratkim opisom sadržaja istih.

Naziv modula	**Kratak opis**
datetime	Sadrži osnovne funkcije za manipuliranje podacima oblika datuma ili vremena
calendar	Sadrži funkcije općenito povezane uz kalendare i nazive dana i mjeseci
math	Sadrži povećani raspon matematičkih funkcija
cmath	Sadrži matematičke funkcije za rad s kompleksnim veličinama
random	Sadrži pseudo-nasumične generatore brojeva za različite oblike raspodjela
statistics	Sadrži matematičke funkcije za statistički račun numeričkih vrijednosti

Tablica 1: Primjer naziva i opisa dostupnih modula u Pythonu

Kao što je navedeno ranije, za pozivanje modula su nam potrebni točni nazivi modula, odnosno ciljanih funkcija sadržanih u njima. Također, treba naglasiti da nakon što se pozove modul, svaki put kada se u nastavku koda koristi funkcija iz modula mora se dodatno uz naziv funkcije navesti iz kojeg modula se poziva ta funkcija. S obzirom na to da se pri optimizaciji koda teži što preglednijem rasporedu tekstualnih dijelova koda, a ispisivanje potpunih naziva modula i funkcija svakih nekoliko redaka koda narušava upravo taj cilj, osigurana je mogućnost pojednostavljenja procesa pozivanja modula u vidu skraćenih naziva ili kratica (eng. *alias*). Pri prvom pozivu modula, uglavnom na samom početku koda, definiramo kojom skraćenicom želimo zamijeniti potpuni naziv modula ili funkcije, tako da kasnije napisani kod bude mnogo pregledniji i jasniji za čitanje, bilo onome tko je pisao kod, a posebno ako netko drugi pregledava kod i želi shvatiti kojim putem je autor koda pokušao doći do rješenja problema. Na slici ispod možemo vidjeti kako izgleda primjer pozivanja modula, odnosno posebno odabranih funkcija iz nekog modula, te kako se vrlo jednostavno implementiraju kratice za nazive.

```
import math

from random import randint

import datetime as dt
```

Slika 5: Primjer pozivanja modula, specifične funkcije iz modula te pozivanja modula pod skraćenim nazivom

U ovom poglavlju su bili opisani moduli dostupni u standardnoj inačici Pythona, no za analizu složenijih fizikalnih sustava i vizualizaciju istih, potrebne su malo složenije funkcije koje su također već definirane, no one su implementirane kao dijelovi zasebnih modula koji nisu dio standardne biblioteke modula u Pythonu. U tom slučaju potrebno je na mrežnim stranicama pronaći datoteke s potrebnim modulima i ručno ih instalirati, to jest povezati s postojećom instalacijom Pythona na računalu. Kako se za vizualizaciju koristi grafika, bilo u dvije dimenzije ili u nekim slučajevima tri dimenzije, potreban je modul koji podržava iscrtavanje grafova, te su potrebni moduli koji mogu upravljati složenijim matematičkim strukturama i oblicima podataka, te u kojima su definirane određene numeričke metode. U ovom radu, za te potrebe se koriste samo tri modula, a to su *„NumPy“*, *„MatPlotLib“* i *„SciPy“*. U nastavku rada navedeni su samo skraćeni opisi navedenih modula koji nisu dio standardne inačice Pythona.

Numerical Python - NumPy

Jedna od glavnih karakteristika ovog modula je povećanje mogućnosti u radu s nizovima za koje se može reći da su središte oko kojeg se vrti ostatak struktura i funkcija ovog paketa. Kao dio modula, osim osnovnih logičkih operatora i aritmetičkih funkcija, dolaze i alati za stvaranje i upravljanje nizovima poput indeksiranja elemenata i sortiranje različitih oblika. Također, podržane su i trigonometrijske, eksponencijalne i logaritamske funkcije, statističke metode i generatori nasumičnih brojeva. Neki dijelovi modula u vidu sadržanih funkcija se preklapaju s drugim modulima, tako da korisnik treba samostalno odabrati koji modul koristiti ovisno o potrebama i trenutnoj situaciji, kao na primjer određeni postupci iz područja linearne algebre i rada s matricama, koji su također uključeni u modulu *ScyPy*. [3]

Scientific Python - ScyPy

ScyPy također sadrži širok raspon matematičkih funkcija i numeričkih metoda, no za rad s istima uglavnom koristi nizove koji su predviđeni kao dio prethodno spomenutog modula *NumPy* tako da je uglavnom preporučeno pozvati oba modula pri pisanju koda. Numeričke metode sadržane u modulu *ScyPy* uglavnom se koriste za numeričko rješavanje diferencijalnih jednadžbi i prilagođavanje krivulja (eng. *curve fitting*) te na metode iz područja linearne algebre. [3]

Primjer unaprijed definiranih funkcija sadržanih u spomenutom modulu su metode za rješavanje sustava linearnih jednadžbi. Naravno, korisnicima je omogućeno definirati vlastite metode ako se želi raditi određenim postupkom, no uobičajeni pristup radi na principu primjene linearne algebre, gdje se sustav linearnih jednadžbi prikazuje u matričnom obliku. Treba naglasiti da Python nema unaprijed ugrađenu strukturu podataka za matrice, već se koristi najjednostavniji način koji zorno prikazuje izgled matrice u matematici ili linearnoj algebri, a to su upravo nizovi, odnosno točnije definirano, koristi se „niz nizova“. U takvoj strukturi svaki element glavnog ili matičnog niza je još jedan, ugniježđeni niz, dok svaki ugniježđeni niz sadrži elemente pojedinih redaka matrice. Način rješavanja sustava linearnih jednadžbi koristeći unaprijed definirane metode uključene u modul *ScyPy* prikazan je na primjeru u nastavku rada.

Neka je zadan sljedeći sustav linearnih jednadžbi :

$$16x_1 + 16x_2 + 17x_3 = 10$$

$$-14x_1 + 17x_2 - 3x_3 = 75$$

$$-5x_1 - 11x_2 - 18x_3 = 43$$

Prvo, kako bi Python prepoznao elemente potrebno je sustav zapisati u matričnom obliku, uz dozvoljene proizvoljne nazive varijabli, odnosno matrica. U ovom primjeru, neka su matrice i varijable označene na sljedeći način, te neka vrijedi jednakost naznačena ispod.

$$A = \begin{bmatrix} 16 & 16 & 17 \\ -14 & 17 & -3 \\ -5 & -11 & -18 \end{bmatrix}, \quad X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, B = \begin{bmatrix} 10 \\ 75 \\ 43 \end{bmatrix}$$

$$A\,X = B$$

```
import numpy as np
import scipy.linalg as linalg

A = np.array([[16, 16, 17], [-14, 17, -3], [-5, -11, -18]])

B = np.array([10, 75, 43])

X = linalg.solve(A, B)
print("Matrica rješenja X =", X)
```

Slika 6: Python kod za rješavanje sustava linearnih jednadžbi

```
Matrica rješenja X = [ 2.  5. -6.]
```

Slika 7: Rezultat pokretanja Python koda za rješavanje sustava linearnih jednadžbi

U Python kodu na slici gore jasno je vidljiv način zapisa matrice koeficijenata A, kako je ranije spomenuto, u obliku glavnog niza i ugniježđenih nizova s elementima pojedinih redaka matrice. Također, vidimo da su matrice B i X zapisane kao jednostavan niz, zato što svaki element niza predstavlja jedan redak, dok u navedenim matricama svaki redak ima samo jedan element, stoga nisu potrebni ugniježđeni nizovi, već je jednostavniji zapis u ovom obliku.

Razmatranjem ovakvog primjera jasno su vidljive mogućnosti primjene ovog modula i uključenih metoda u bilo kojem području gdje se koriste matrice za prikazivanje podataka, kao na primjer određivanje svojstvenih funkcija odnosno svojstvenih vrijednosti matrica i slično.

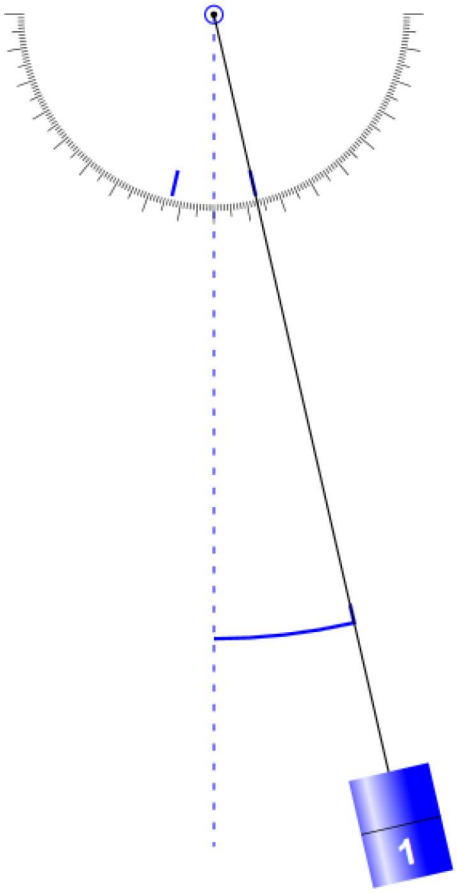
Mathematical Plotting Library - MatPlotLib

MatPlotLib je standardni modul u Pythonu za stvaranje dvodimenzijskih i trodimenzijskih grafika i prikaza. U ovom se modulu također za jednostavniji rad i organizaciju podataka aktivno i učinkovito koriste nizovi definirani u modulu *NumPy* tako da je i u ovom slučaju preporučeno koristiti module istovremeno.[3] Naravno, kombiniranje modula ne predstavlja nikakve poteškoće jer sve što je potrebno napraviti jest pozvati module pri pisanju koda i sve je spremno za daljnje pisanje koda.

Primjeri primjene metoda uključenih u modul *MatPlotLib* vidljivi su u pokaznom primjeru vizualizacije problema iz fizike u nastavku rada, odnosno u Python kodu za pokazni primjer koji se nalazi u dodatku na kraju rada.

Primjer – matematičko njihalo

Načini primjene i mogućnosti rada s računalom u svrhu analize i vizualizacije problema iz fizike može se prikazati na brojnim primjerima, poput problema matematičkog njihala. U ovom poglavlju rada bit će pokazano kako pomoću Pythona i dodanih modula u Pythonu prvo zapisati jednadžbe koje opisuju stanje promatranog sustava, a zatim kako upravljati njima i na kraju kako koristeći Python nacrtati krivulje energije promatranog sustava. Treba naglasiti da se u ovom pokaznom primjeru u obzir uzimaju određene aproksimacije radi pojednostavljenja problema. Neke od važnijih aproksimacija jesu vezane za ključne dijelove sustava, gdje se pretpostavlja da je sva masa tijela homogeno sažeta u materijalnu točku koja je ovješena na jedan kraj homogenog jednodimenzionalnog štapa nepromjenjive duljine, dok je drugi kraj nepomično učvršćen u ovjesnoj točki. Također, pretpostavlja se da sustav titra u ravnini, te da na njega nemaju nikakvog utjecaja vanjska gibanja, poput rotacije Zemlje ili slično.



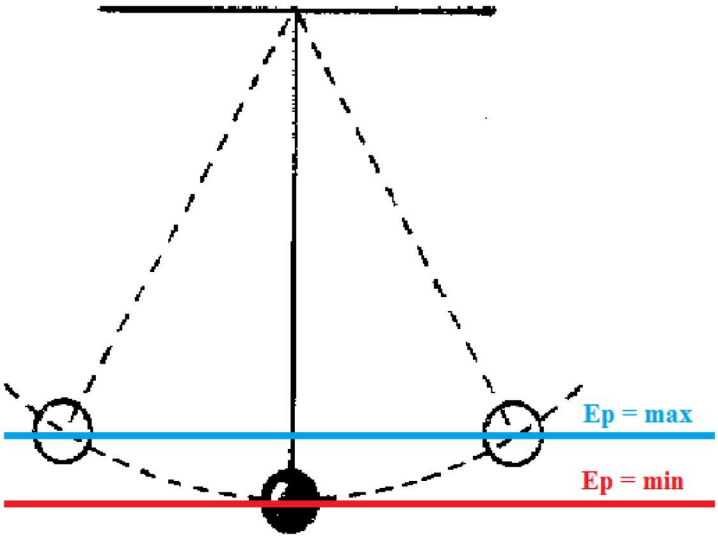
Slika 7: Prikaz matematičkog njihala

Gibanje njihala se postiže tako da se uloži rad kako bi se uteg izvukao van položaja ravnoteže, te se otpusti nakon čega počinje titranje. Ovdje se također uvodi pretpostavka da je matematičko njihalo slobodno, odnosno ne postoje faktori prigušivanja ili ubrzavanja koji bi utjecali na gibanje njihala, te se pretpostavlja da je u trenutku početka gibanja uteg otpušten bez dodavanja početnog ubrzanja. Ako se uvedu i aproksimacije za malen kut početnog otklona, sustav možemo početi promatrati poput harmonijskog oscilatora, što znatno pojednostavljuje promatrani sustav jer su jednadžbe gibanja harmonijskog oscilatora i njihova rješenja opće poznata iz klasične mehanike. Kako bi se moglo nacrtati krivulju ukupne energije matematičkog njihala, potrebno je moći odrediti kut otklona njihala u odnosu na početni položaj u bilo kojem trenutku, kako bi se za taj trenutak mogla odrediti energija i označiti na krivulji. Upravo iz tog razloga su potrebne određene jednadžbe iz klasične mehanike.

Neka vrijedi $-\pi \leq \varphi \leq \pi$, gdje je φ kut otklona njihala u odnosu na ravnotežni položaj, te neka su dani početni otklon njihala φ_0 i početna brzina njihala $\dot{\varphi}_0$, uz naglasak da se radi o kutovima zadanim u radijanima. Pošto je uvedena pretpostavka da sustav titra u ravnini, logično je sustav promatrati u polarnom koordinatnom sustavu. Raspisivanjem jednadžbe gibanja u polarnom koordinatnom sustavu pomoću vektora položaja materijalne točke, odnosno utega u trenutku *t* i sila koje djeluju na njega, dobije se jednadžba oblika $\ddot{\varphi} + \frac{g}{l} \cdot \sin\varphi = 0$, što se uz navedenu aproksimaciju za malene kutove može prepoznati kao jednadžba slobodnog jednodimenzijskog harmonijskog oscilatora. Razvojem dobivene jednadžbe poput općeg rješenja jednadžbe gibanja harmonijskog oscilatora, te uvrštavanjem početnih uvjeta $\varphi(0) = \varphi_0$ i $\dot{\varphi(0)} = 0$, za kut otklona u trenutku *t* dobije se jednadžba oblika :

$$\varphi(t) = \varphi_0 \cdot \cos\left[\sqrt{\frac{g}{l}} \cdot t\right].$$

Nadalje, potrebno je definirati izraz pomoću kojeg će se odrediti energija sustava u svakom trenutku. Kako bi to bilo moguće, potrebno je odrediti referentnu točku u odnosu na koju će biti promatran sustav. U ovom slučaju, sustav je definiran tako da je najniža točka utega, odnosno položaj ravnoteže, ujedno i točka najmanje potencijalne energije utega.



Slika 8: Prikaz potencijalne energije u ravnotežnom položaju u odnosu na rubne položaje

U promatranom sustavu poznati su početni otklon u odnosu na ravnotežni položaj, masa utega, duljina štapa odnosno niti i gravitacijska konstanta, te neka je vertikalna razlika u visini između ravnotežnog i rubnog položaja označena s *h*, tada će potencijalna energija utega u nekom trenutku *t* biti dana izrazom $E_p(t) = m \cdot g \cdot h$. Primjene li se pravila trigonometrije, moguće je izraziti nepoznatu vertikalnu razliku u visini *h* pomoću poznatih duljine štapa i kuta otklona njihala, nakon čega gornji izraz za potencijalnu energiju poprima oblik :

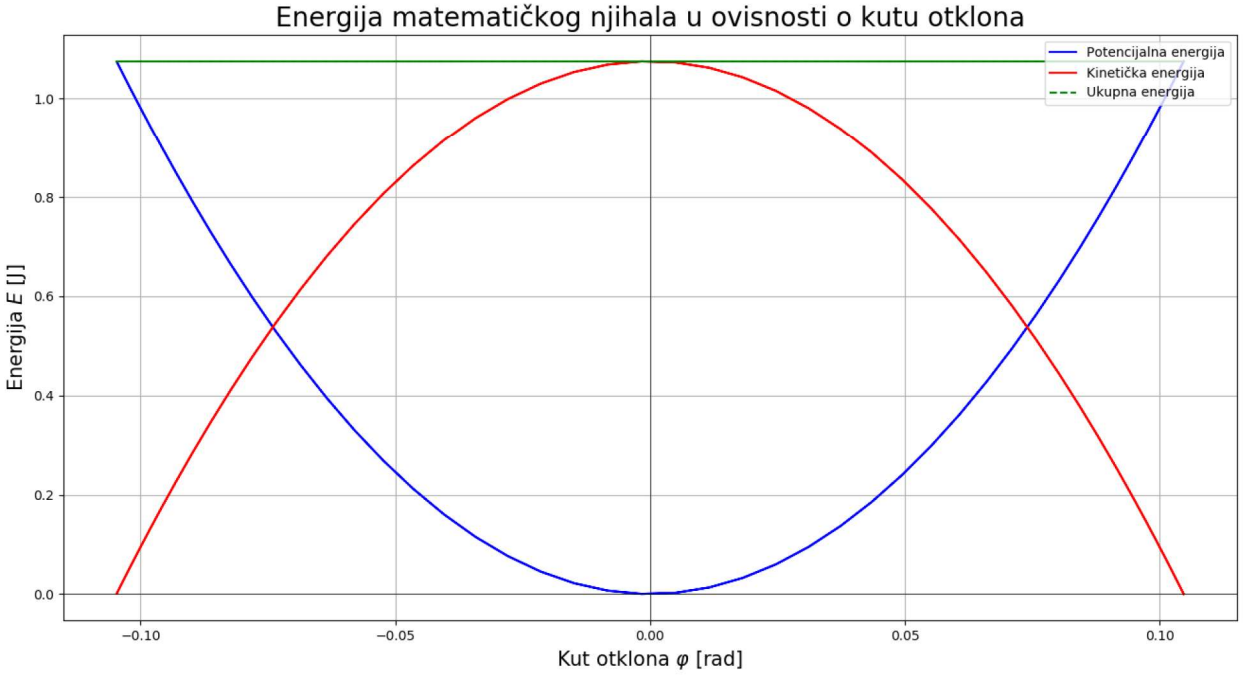
$$E_p(t) = m \cdot g \cdot l \cdot (1 - \cos \varphi(t))$$

Ključni dio promatranja ovog sustava pomoću Pythona je upravo zapisivanje ovih jednadžbi u sintaksi programskog jezika. Koristeći zakone očuvanja, u ovom slučaju zakon očuvanja energije, uz poznavanje potencijalne energije u rubnim položajima, vrlo je lako odrediti ukupnu energiju. Nakon toga, zbog simetrije promatranog sustava, ponovno prijelazom iz pravokutnog koordinatnog sustava na promatranje njihala u polarnom koordinatnom sustavu može se odrediti izraz za kinetičku energiju njihala.

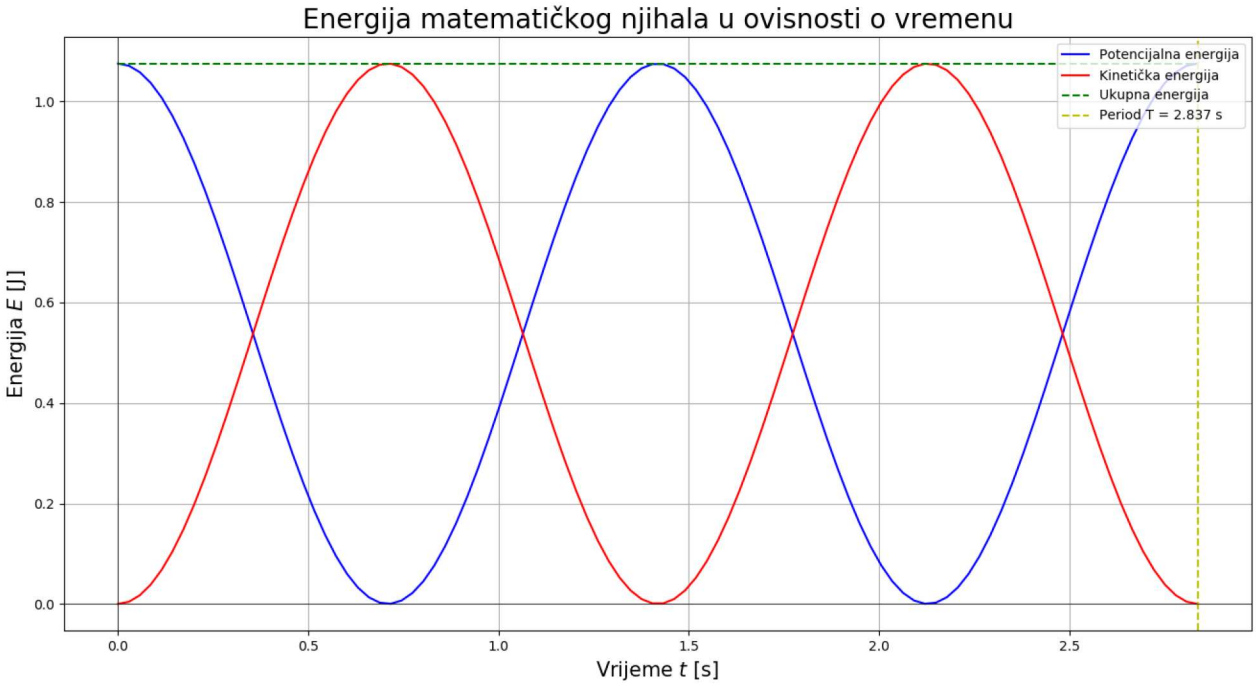
$$E_{UK} = E_P(t = 0)$$

$$E_K = \frac{1}{2} \cdot m \cdot v^2 \rightarrow E_K = \frac{1}{2} \cdot m \cdot l^2 \cdot \dot{\varphi}^2$$

Primjenom metoda koje su unaprijed definirane i nalaze se u dodatnim modulima koji su ranije opisani, možemo dobiti krivulje energije promatranog sustava, kao što je prikazano na slikama ispod gdje su nacrtane krivulje ovisnosti energije o vremenu i ovisnosti energije o kutu otklona od ravnotežnog položaja za vrijeme jednog perioda titranja, uz vrijednosti $m_{uteg} = 10\ \text{kg}$; $g = 9.81\ {}^{\text{m}}/_{\text{s}^2}$; $l_{nit} = 2\ \text{m}$; $\varphi_0 = 6°$. Treba naglasiti da je zadan otklon u stupnjevima iako je gore u pripremi navedeno da otklon promatramo u radijanima. Radi jednostavnosti unosa početni kut otklona se zadaje u stupnjevima, a u kodu je napisana pretvorba iz stupnjeva u radijane. Potpuni kod kojim je nacrtana krivulja nalazi se u dodatku na kraju rada, no treba naglasiti da se crtanje obje krivulje izvodi korištenjem jednog koda, uz promjenu jednog bloka naredbi koji je jasno naznačen.



Slika 9: Primjer krivulje ovisnosti energije o kutu otklona za matematičko njihalo



Slika 10: Primjer krivulje ovisnosti energije o proteklom vremenu za matematičko njihalo

Zaključak

Kada bi se pokušalo razmotriti u koje se svrhe sve mogu primjenjivati alati poput računala u znanosti, neovisno o kojem području govorili, vrlo brzo bi postalo jasno da mogućnostima gotovo nema kraja. Naravno, kao student preddiplomskog studija mogu govoriti samo iz relativno ograničenog stvarnog iskustva, no tijekom studija svejedno sam došao u kontakt s problemima za koje je vrlo lako uvidjeti jasne prednosti izvršavanja barem dijela problemskog zadatka pomoću računala. Od grafičkog prikaza raznovrsnih krivulja s kojima se susrećemo u teorijski orijentiranim kolegijima kako bi imali točniji uvid u nešto o čemu se inače samo teorijski raspravlja, sve do primjene programiranja za lakšu obradu većih skupova numeričkih podataka iz kolegija praktične prirode, za izvođenje određene vrste statističke obrade ili sakupljanja podataka u smislene cjeline.

Treba uzeti u obzir i potencijalnu složenost zadatka, tako da je primjena programskog jezika poput Pythona, za kojeg se može reći da je donekle uravnotežen, vrlo razuman izbor. Usporedimo li s jedne strane skup mogućnosti koje pruža njegova primjena, a s druge složenost njegove praktične primjene u smislu sintakse i dostupnosti priručnika i dodatnih paketa, Python se ispostavlja kao vrlo dobar izbor, pogotovo zato što je besplatan i svima dostupan, bilo samo osnovna inačica ili u sklopu s dodatnim paketima. Uzimajući u obzir sve veći opseg mogućnosti koji dolazi s novim inačicama programskih jezika, i promatrajući trend njihove primjene u bližoj prošlosti, vrlo je lako predvidjeti da će primjena računala, a samim time i programiranja, za analizu, vizualizaciju i na posljetku obradu podataka postati sve veća i veća, može se reći gotovo paralelno s porastom količine podataka koja se prikuplja iz dana u dan.

Literatura

[1] Landau, R. H.; Jose Paez, M.; Bordeianu, C. C., Computational Physics : Problem Solving with Computers, 2007.

[2] Ayars, E., Computational Physics with Python, 2013.

[3] Pine, D. J., Introduction to Python for Science and Engineering, CRC Press, 2019.

[4] Glumac, Z., Klasična Mehanika : Kratak Uvod, 2015. (posljednja verzija preuzeta 2.9.2019., dostupno na http://gama.fizika.unios.hr/~zglumac/utm.pdf)

[5] Horvat, D.; Hrupec, D., Fizika 1 : Udžbenik za 1. razred gimnazija, Element, 2019.

Poveznice

[6] https://www.python.org/

[7] https://www.numpy.org/

[8] https://www.scipy.org/

[9] https://matplotlib.org/

[10] https://phet.colorado.edu/sims/html/pendulum-lab/latest/pendulum-lab_en.html

Životopis

Kristijan Dragičević rođen je 25. listopada 1997. godine u Vinkovcima. Osnovnu školu završio je u Starim Mikanovcima, a zatim Tehničku školu Ruđera Boškovića Vinkovci u Vinkovcima. Trenutno studira fiziku na Odjelu za fiziku Sveučilišta Josipa Jurja Strossmayera u Osijeku.

Dodatak

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

masa = float(input("Molimo unesite masu tijela ovješenog o njihalo u kilogramima : "))
duljina = float(input("Molimo unesite duljinu niti njihala u metrima : "))
otklon = float(input("Molimo unesite otklon njihala u stupnjevima u odnosu na položaj mirovanja : "))
g = 9.81 # m / s^2

omega = np.sqrt(g / duljina)
T = 2 * np.pi * np.sqrt(duljina / g)
kut0 = np.deg2rad(otklon)
t = np.linspace(0, T, 100)
kut = kut0 * np.cos(omega * t)

en_k = 0.5 * masa * g * duljina * (kut0**2 - kut**2)
en_p = 0.5 * masa * g * duljina * kut**2
en_uk = en_k + en_p

# krivulja ovisnosti energije o kutu otklona
plt.plot(kut, en_p, "b-", label="Potencijalna energija")
plt.plot(kut, en_k, "r-", label="Kinetička energija")
plt.plot(kut, en_uk, "g--", label="Ukupna energija")
plt.xlabel("Kut otklona " +r"$\varphi$" + " [rad]", size = 15)
plt.title("Energija matematičkog njihala u ovisnosti o kutu otklona", size = 20)

# krivulja ovisnosti energije o proteklom vremenu
#plt.plot(t, en_p, "b-", label="Potencijalna energija")
#plt.plot(t, en_k, "r-", label="Kinetička energija")
#plt.plot(t, en_uk, "g--", label="Ukupna energija")
#plt.axvline(T, 0, 1, c="y", ls="--", label="Period T = " + str(round(T, 3)) + " s")
#plt.xlabel("Vrijeme " +r"$t$" + " [s]", size = 15)
#plt.title("Energija matematičkog njihala u ovisnosti o vremenu", size = 20)

plt.ylabel("Energija " +r"$E$" + " [J]", size = 15)
plt.grid()
plt.axvline(color="black", linewidth = 0.5)
plt.axhline(color="black", linewidth = 0.5)
plt.legend(loc="upper right")
plt.show()
```

Dodatak 1 : Python kod pokaznog primjera za matematičko njihalo

Primjeri analize i vizualizacije zadataka iz odabranog srednjoškolskog udžbenika fizike

Sljedećih nekoliko primjera su stvarni zadaci preuzeti iz srednjoškolskog udžbenika fizike [5], iz područja gibanja tijela. Rješenja su grafički prikazana pomoću programskog jezika Python u dodatku ispod. Na ovako jednostavnim primjerima može se vidjeti da ovakav način prikaza rješenja pruža veliku razinu preciznosti prikazanih krivulja u odnosu na zadane numeričke podatke o promatranim fizikalnim veličinama, naravno uz preduvjet da je Python kod točno napisan.

Primjer 1 – cjelina 1.5, zadatak 20

Automobil vozi brzinom $30\ \mathrm{ms}^{-1}$. U nekom trenutku vozač uoči opasnost i počne naglo kočiti. Vrijeme reakcije vozača je 0.8 s (to je vrijeme od trenutka uočavanja opasnosti do trenutka početka smanjivanja brzine; za to se vrijeme brzina automobila ne mijenja). Nakon toga automobil počinje usporavati $a = -7.5\ \mathrm{ms}^{-2}$, do zaustavljanja. Nacrtajte graf brzine automobila *v(t)* . Koliki je ukupni prijeđeni put automobila od trenutka uočavanja opasnosti do zaustavljanja?

Rješenje

Promatrani zadatak može se podijeliti na dva dijela, odnosno dva vremenska perioda u kojima se automobil različito giba. U prvom dijelu, koji traje koliko i vrijeme reakcije vozača, automobil se giba jednoliko pravocrtno, i nakon toga, u drugom dijelu automobil se giba jednoliko usporeno sve do zaustavljanja. Kako bi se mogao nacrtati graf ovisnosti brzine o vremenu, potrebno je poznavati koliko ukupno vremena traje promatrano gibanje, što će se dobiti zbrajanjem vremena reakcije vozača i vremena kočenja. Traženo vrijeme kočenja može se odrediti iz početne brzine i poznate akceleracije. Krivulju ovisnosti puta o vremenu se također može podijeliti na dva dijela, put pređen tijekom pravocrtnog gibanja, i put pređen tijekom kočenja, dok će ukupni put biti njihov zbroj. Konačne krivulje *v(t)* i *s(t)* dobiti će se tako da pojedine dijelove definiramo zasebno, a zatim ih pomoću Python koda nacrtamo jedne za drugima, sa potrebnim pomacima po koordinatnim osima, kao što je vidljivo na slici koda ispod. U nastavku se može vidjeti postupak određivanja traženih veličina, te jednadžbe pomoću kojih su nacrtane tražene krivulje.

$$t_{reakcija} = 0.8\ \mathrm{s}$$

$$v_{poč} = 30\ \mathrm{ms}^{-1}$$

$$a_{kočenje} = -7.5\ \mathrm{ms}^{-2}$$

$$t_{kočenje} = \left| \frac{a_{kočenje}}{v_{poč}} \right| = \left| \frac{-7.5\ \mathrm{ms}^{-2}}{30\ \mathrm{ms}^{-1}} \right| = 4\ \mathrm{s}$$

$$t_{ukupno} = t_{reakcija} + t_{kočenje} = 0.8\ \mathrm{s} + 4\ \mathrm{s} = 4.8\ \mathrm{s}$$

$$s_1 = v_{poč} \cdot t_{reakcije} = 30\ \mathrm{ms}^{-1} \cdot 0.8\ \mathrm{s} = 24\ \mathrm{m}$$

$$s_2 = v_{poč} \cdot t_{kočenje} + \frac{1}{2} \cdot a_{kočenje} \cdot t_{kočenje}^2 = 60\ \mathrm{m}$$

$$s_{uk} = s_1 + s_2 = 60\ \mathrm{m} + 24\ \mathrm{m} = 84\ \mathrm{m}$$

$$v_1(t) = konst. = v_{poč}$$

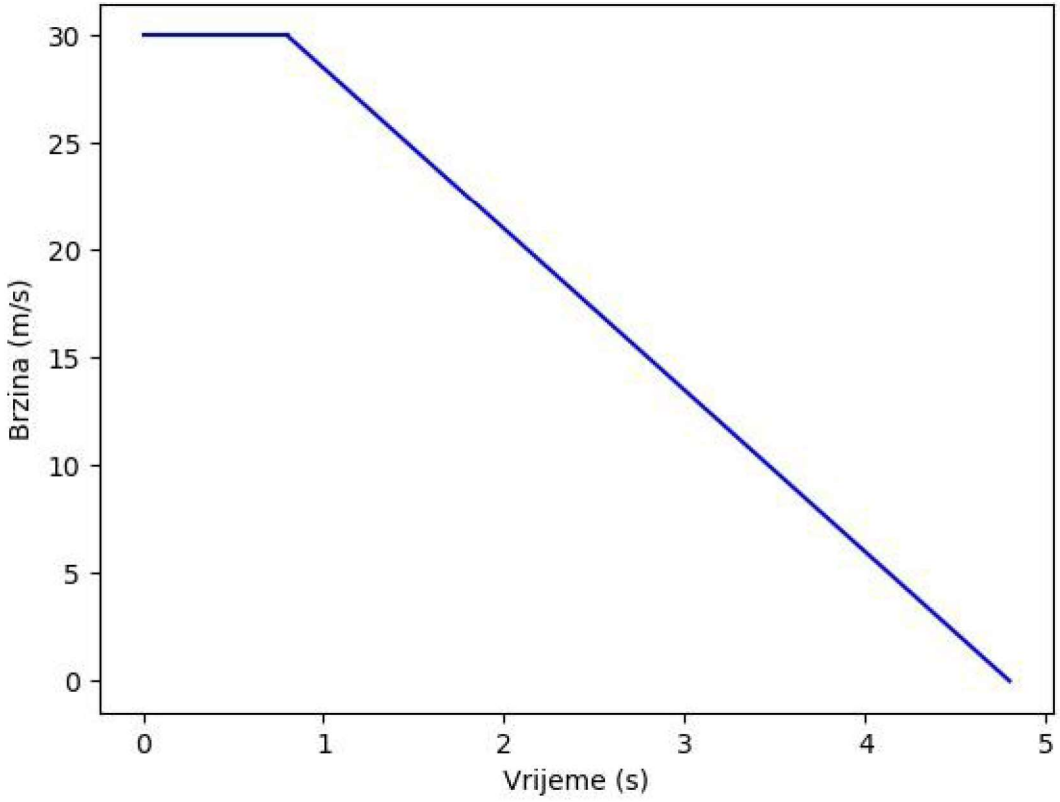
$$v_2(t) = v_{poč} + a_{kočenje} \cdot t$$

```
#cjelina 1.5 - zadatak 20
import numpy as np
import matplotlib.pyplot as plt

#vrijednosti zadane u tekstu zadatka
pocetna_brzina = 30
vrijeme_reakcije = 0.8
akceleracija = -7.5

#definiramo funkcije za dvije vrste gibanja koje se javljaju u zadatku
def jednoliko_pravocrtno(vrijeme):
    brzina = pocetna_brzina
    put = brzina * vrijeme
    return brzina, put

def jednoliko_usporeno(vrijeme):
    brzina = pocetna_brzina + ( akceleracija * vrijeme )
    put = pocetna_brzina * vrijeme + 0.5 * akceleracija * vrijeme * vrijeme
    return brzina, put

#definiramo periode za pojedino gibanje
vrijeme_zaustavljanja = abs(pocetna_brzina / akceleracija)
vrijeme_1 = np.linspace(0,vrijeme_reakcije,100)
vrijeme_2 = np.linspace(0,vrijeme_zaustavljanja,100)

#definiramo nizove za crtanje v(t) i s(t) grafova
v1 , s1 = jednoliko_pravocrtno(vrijeme_1)
v2 , s2 = jednoliko_usporeno(vrijeme_2)

v1_niz = v1 * np.ones(100)
put_2 = s2 + pocetna_brzina * vrijeme_reakcije
put_uk = str("Ukupni prijeđeni put je " + str(put_2[-1]) + " metara")
```

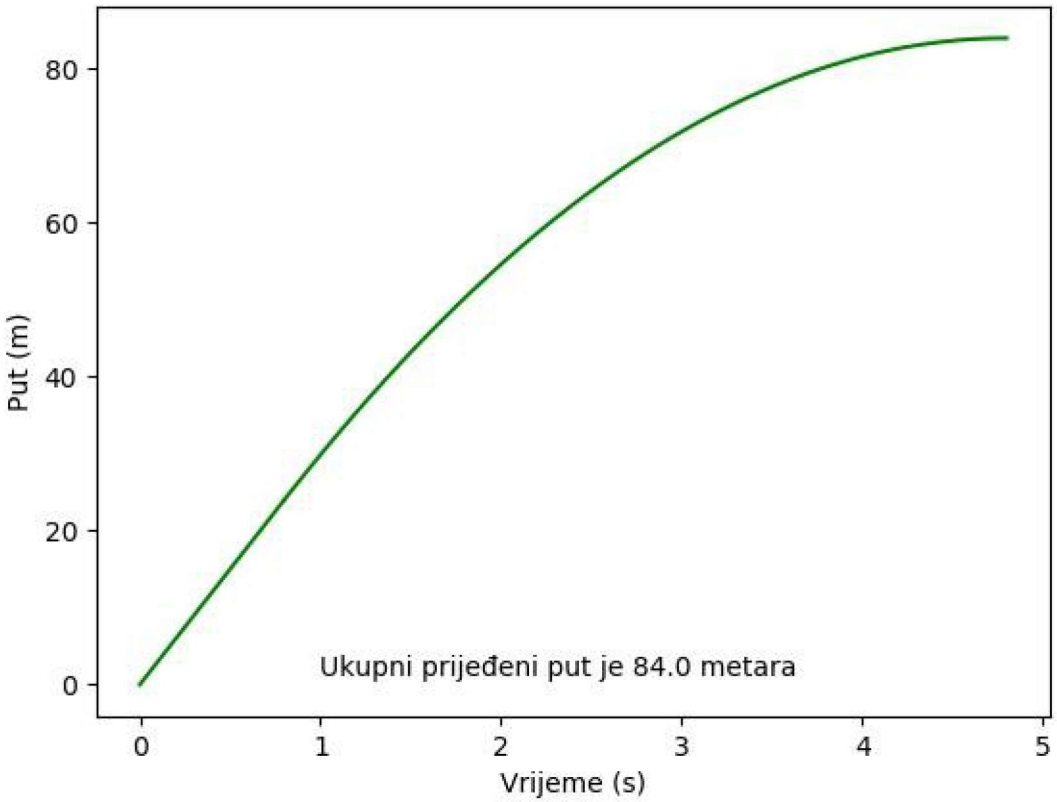
Dodatak 2: Python kod za primjer 1, 1/2

```
#crtamo grafove
plt.plot(vrijeme_1 , v1_niz , "b-")
plt.plot(vrijeme_2 + vrijeme_reakcije , v2 , "b-")
plt.xlabel("Vrijeme (s)")
plt.ylabel("Brzina (m/s)")
plt.show()

plt.plot(vrijeme_1 , s1 ,"g-")
plt.plot(vrijeme_2 + vrijeme_reakcije , put_2 , "g-")
plt.text(1,1,put_uk)
plt.xlabel("Vrijeme (s)")
plt.ylabel("Put (m)")
plt.show()
```

Dodatak 3: Python kod za primjer 1, 2/2



Dodatak 4: Krivulja ovisnosti brzine o vremenu za primjer 1



Dodatak 5: Krivulja ovisnosti puta o vremenu za primjer 1

Primjer 2 – cjelina 1.5, zadatak 14

Automobil vozi brzinom 36 $\mathrm{kmh^{-1}}$. U nekom trenutku počinje ubrzavati konstantnim ubrzanjem, tako da za 10 s postigne brzinu 30 $\mathrm{ms^{-1}}$. Skicirajte ovisnost brzine o vremenu za prvih 10 s gibanja. Koliki put automobil prijeđe u prvih 6 s od početka ubrzavanja?

Rješenje

Za početak, potrebno je pretvoriti sve poznate fizikalne veličine u iste mjerne jedinice kako bi mogli baratati njima. U ovom se zadatku radi samo o jednoj vrsti gibanja, tako da će crtanje krivulje i rješavanje zadatka biti jednostavnije. Opet je potrebno uzeti u obzir podatke o početnim uvjetima iz kojih se može odrediti kako izgleda gibanje nakon nekog vremena, u ovom slučaju zadanog perioda od 10 sekundi.

$$v_{poč} = 36\ \mathrm{kmh^{-1}} = 36 \cdot \frac{1000\ \mathrm{m}}{3600\ \mathrm{s}} = 10\ \mathrm{ms^{-1}}$$

$$v_{kon} = 30\ \mathrm{ms^{-1}}$$

$$t_{ubrzavanje} = 10\ \mathrm{s}$$

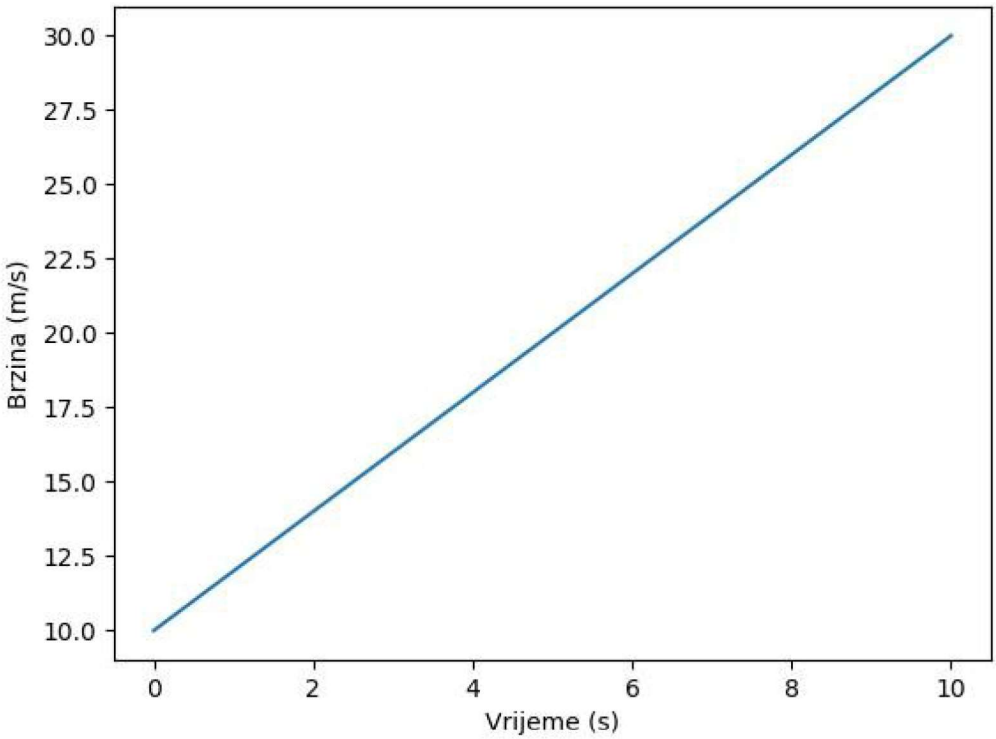
$$t_1 = 6\ \mathrm{s}$$

$$v(t) = v_{poč} + \left(v_{kon} - v_{poč}\right) \cdot t$$

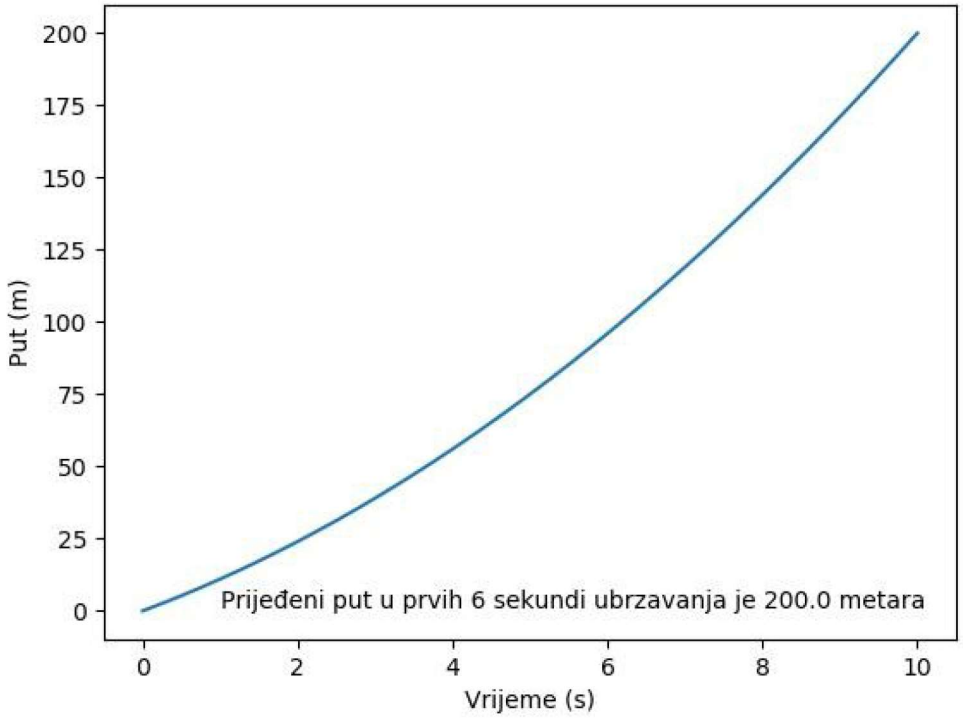
$$s = v_{poč} \cdot t_1 + \frac{1}{2} \cdot \left(v_{kon} - v_{poč}\right) \cdot t_1^2$$

```python
#cjelina 1.5, zadatak 14
import numpy as np
import matplotlib.pyplot as plt

def jednoliko_ubrzano(vrijeme):
    pocetna_brzina = 36 * ( 1000 / 3600 )
    konacna_brzina = 30
    vrijeme_ubrzavanja = 10
    akceleracija = (konacna_brzina - pocetna_brzina) / vrijeme_ubrzavanja
    brzina = pocetna_brzina + akceleracija * vrijeme
    put = pocetna_brzina * vrijeme + 0.5 * akceleracija * vrijeme * vrijeme
    return brzina, put

t = np.linspace(0,10,200)

v, s = jednoliko_ubrzano(t)
put_1 = str("Prijeđeni put u prvih 6 sekundi ubrzavanja je "+str(s[-1])+" metara")

plt.plot(t,v)
plt.xlabel("Vrijeme (s)")
plt.ylabel("Brzina (m/s)")
plt.show()

plt.plot(t,s)
plt.xlabel("Vrijeme (s)")
plt.ylabel("Put (m)")
plt.text(1,1,put_1)
plt.show()
```

Dodatak 6: Python kod za primjer 2



Dodatak 7: Krivulja ovisnosti brzine o vremenu za primjer 2



Dodatak 8: Krivulja ovisnosti puta o vremenu za primjer 2

Primjer 3 – cjelina 1.5, zadatak 21

Dvije tramvajske stanice udaljene su 450 m. Prvih 10 s tramvaj se pri polasku iz jedne stanice giba jednoliko ubrzano, a zatim nastavlja gibanje stalnom brzinom od 36 kmh^{-1} . Posljednjih 10 s pred drugom stanicom giba se jednoliko usporeno, do zaustavljanja. Nacrtajte graf brzine tramvaja *v(t)* i izračunajte put koji je tramvaj prošao gibajući se stalnom brzinom.

Rješenje

Ovaj primjer također treba podijeliti na segmente, u ovom slučaju na tri različite vrste gibanja. U ovom slučaju poznajemo ukupan put i poznajemo trajanje prvog i posljednjeg segmenta, a potrebno je odrediti put koji je tramvaj prošao tijekom drugog segmenta, što je moguće odrediti oduzimanjem prijeđenog puta u prvom i posljednjem segmentu od ukupnog puta.

$$s_{uk} = 450\ \mathrm{m}$$

$$v_2 = 36\ \mathrm{kmh}^{-1} = \ 10\ \mathrm{ms}^{-1}$$

$$t_1 = t_3 = 10\ \mathrm{s}$$

$$a_1 = \frac{v_2 - 0}{t_1} = \ 1\ \mathrm{ms}^{-2}$$

$$a_3 = \ \frac{0 - v_2}{t_3} = \ -1\ \mathrm{ms}^{-2}$$

$$s_1 = \frac{1}{2} \cdot a_1 \cdot t_1^2 = 50\ \mathrm{m}$$

$$s_3 = v_2 \cdot t_3 + \frac{1}{2} a_3 \cdot t_3^2 = 50\ \mathrm{m}$$

$$s_2 = s_{uk} - s_1 - s_3 = 450 - 50 - 50 = 350\ \mathrm{m}$$

$$v_1(t) = a_1 \cdot t$$

$$v_2(t) = konst. = v_2$$

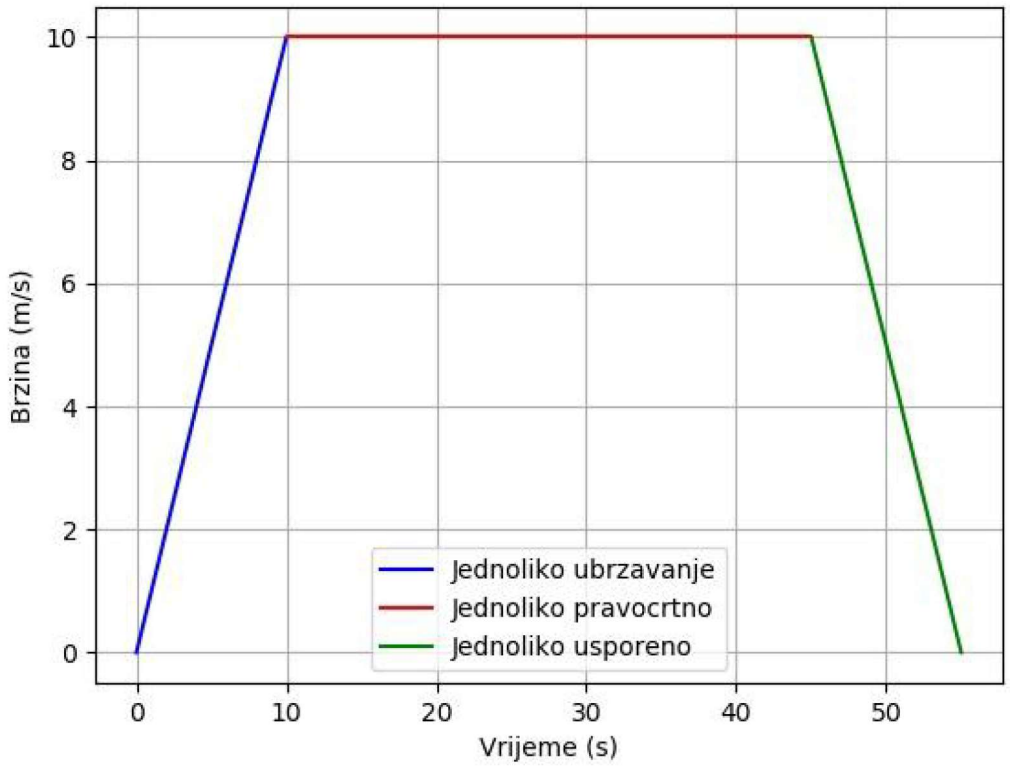
$$v_3(t) = a_3 \cdot t$$

```
#cjelina 1.5, zadatak 21
import numpy as np
import matplotlib.pyplot as plt

def jednoliko_ubrzano(vrijeme):
    pocetna_brzina = 0
    konacna_brzina = 36 * ( 1000 / 3600 )
    vrijeme_ubrzavanja = 10
    akceleracija = ( konacna_brzina - pocetna_brzina ) / vrijeme_ubrzavanja
    put = 0.5 * akceleracija * vrijeme * vrijeme
    brzina = pocetna_brzina + akceleracija * vrijeme
    return brzina,put,akceleracija

def jednoliko_pravocrtno(vrijeme):
    brzina = 36 * ( 1000 / 3600 ) * np.zeros(200)
    put = brzina * vrijeme
    akceleracija = 0
    return brzina,put,akceleracija

def jednoliko_usporeno(vrijeme):
    pocetna_brzina = 36 * ( 1000 / 3600 )
    konacna_brzina = 0
    vrijeme_usporavanja = 10
    akceleracija = ( konacna_brzina - pocetna_brzina ) / vrijeme_usporavanja
    put = pocetna_brzina * vrijeme + 0.5 * akceleracija * vrijeme * vrijeme
    brzina = pocetna_brzina + akceleracija * vrijeme
    return brzina,put,akceleracija

vrijeme_ubrzavanja = 10
vrijeme_usporavanja = 10
brzina_1 = 36*(1000/3600)
akceleracija_1 = brzina_1 / vrijeme_ubrzavanja
akceleracija_3 = -brzina_1 / vrijeme_usporavanja
put_uk = 450

put_1 = 0.5*akceleracija_1 * vrijeme_ubrzavanja**2
put_3 = brzina_1 * vrijeme_usporavanja + 0.5 * akceleracija_3 * vrijeme_usporavanja**2
vrijeme_2 = (put_uk - put_1 - put_3) / brzina_1
pomak = put_uk - put_3
```

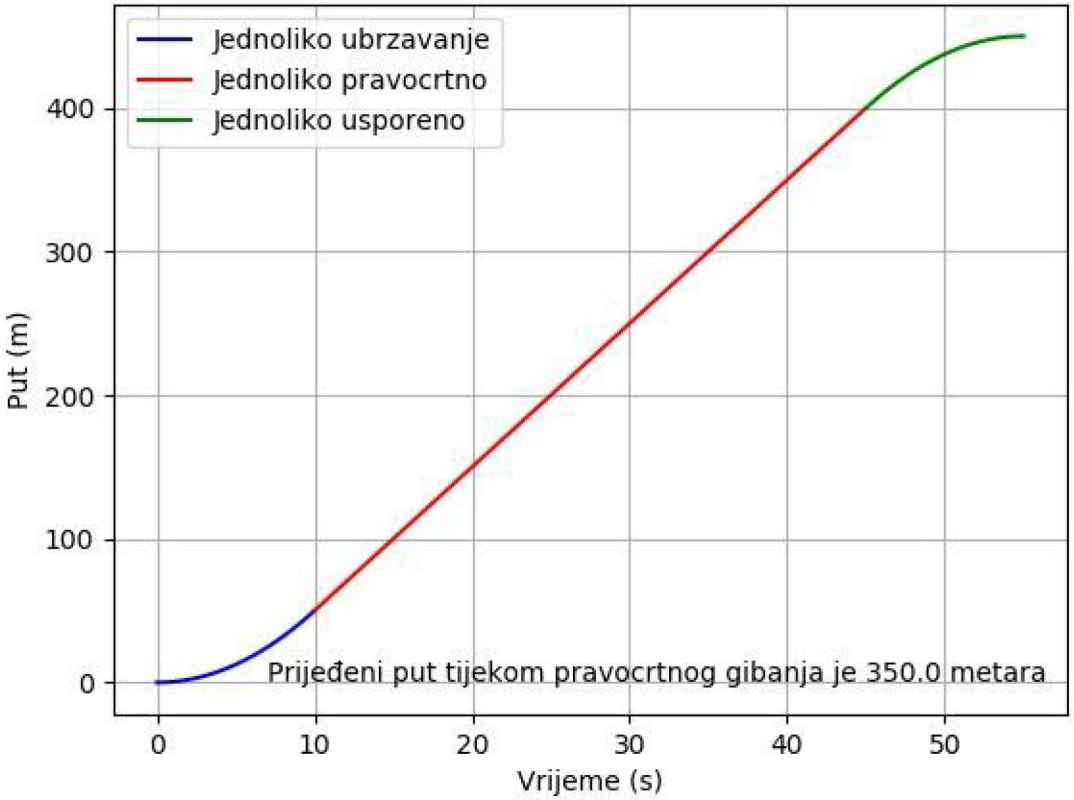
Dodatak 9: Python kod za primjer 3, 1/2

```
t1 = np.linspace(0,vrijeme_ubrzavanja,200)
t2 = np.linspace(0,vrijeme_2,200)
t3 = np.linspace(0,vrijeme_usporavanja,200)

v1,s1,a1 = jednoliko_ubrzano(t1)
v2,s2,a2 = jednoliko_pravocrtno(t2)
v3,s3,a3 = jednoliko_usporeno(t3)
put_2 = brzina_1 * t2
put_2uk = str("Prijeđeni put tijekom pravocrtnog gibanja je "+str(put_2[-1])+" metara")

plt.plot(t1,v1,"b-",label="Jednoliko ubrzavanje")
plt.plot(t2 + vrijeme_ubrzavanja,v2 + brzina_1,"r-",label="Jednoliko pravocrtno")
plt.plot(t3 + vrijeme_ubrzavanja + vrijeme_2,v3,"g-",label="Jednoliko usporeno")

plt.xlabel("Vrijeme (s)")
plt.ylabel("Brzina (m/s)")
plt.legend()
plt.grid()
plt.show()

plt.plot(t1,s1,"b-",label="Jednoliko ubrzavanje")
plt.plot(t2 + vrijeme_ubrzavanja,put_2 + put_1,"r-",label="Jednoliko pravocrtno")
plt.plot(t3 + vrijeme_ubrzavanja + vrijeme_2,s3 + pomak,"g-",label="Jednoliko usporeno")
plt.text(7,1,put_2uk)
plt.xlabel("Vrijeme (s)")
plt.ylabel("Put (m)")
plt.legend()
plt.grid()
plt.show()
```

Dodatak 10: Python kod za primjer 3, 2/2



Dodatak 11: Krivulja ovisnosti brzine o vremenu za primjer 3



Dodatak 12: Krivulja ovisnosti puta o vremenu za primjer 3