Izrada simulacija u teorijskoj fizici pomoću programskog jezika Python

Herceg, Mihovil

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:935833

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-21**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Mihovil Herceg

IZRADA SIMULACIJA U TEORIJSKOJ FIZICI POMOĆU PROGRAMSKOG JEZIKA PYTHON

Diplomski rad

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

INTEGRIRANI PREDDIPLOMSKI I DIPLOMSKI SVEUČILIŠNI STUDIJ
FIZIKA I INFORMATIKA; SMJER: NASTAVNIČKI

Mihovil Herceg

Diplomski rad

Izrada simulacija u teorijskoj fizici pomoću programskog jezika Python

Voditelj diplomskog rada: prof. dr. sc. Tamara Nikšić

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____

Zagreb, 2022.

Sažetak

Programski jezik Python je široko rasprostranjen i ima razne primjene, kako u znanstvenim istraživanjima, tako i u edukaciji. Njegov paket VPython daje alate za laku izradu 3d prikaza i iscrtavanje grafova te ga možemo koristiti za izradu računalnih simulacija. Simulacije možemo koristiti kao zamjenu i nadopunu pokusima posebno kada je pokus zbog prirode istraživane pojave teško ili nemoguće izvesti u učionici. Koristeći VPython, Newtonove zakone gibanja i zakon gravitacije možemo napraviti računalne simulacije planetarnog gibanja, točnije simulaciju gibanja dva tijela koja međudjeluju gravitacijskom silom. Iz tih simulacija možemo prikazati Keplerove zakone koji opisuju orbite planeta oko Sunca. Također se može prikazati gravitacijska praćka, manevar koji koriste međuplanetarne letjelice za promjenu smjera i iznosa brzine. Izrađene simulacije možemo koristiti u nastavi fizike kako bi pružili učenicima interaktivno okruženje u kojemu mogu proučavati Keplerove zakone i gravitacijsku praćku te vođeni nastavnikom doći do zaključaka o gibanjima planeta oko Sunca i gravitacijskoj praćki.

Ključne riječi: Python, VPython, Keplerovi zakoni, gravitacijska praćka, računalne simulacije

Abstract

The Python programming language is widespread and has various applications, both in scientific research and in education. Its VPython package provides tools for easy 3d rendering and graphing so it can be used to create computer simulations, Simulations can be used as a replacement and supplement to experiments, especially when the experiment is difficult or impossible to perform in the classroom due to the nature of the researched phenomenon. Using VPython, Newton's laws of motion and the law of gravity we can make computer simulations of planetary motion, more precisely the simulation of motion of two bodies interacting with gravitational force. From these simulations we can show Kepler's laws describing the orbits of the planets around the Sun. Gravitational slingshot, a maneuver used by interplanetary spacecraft to change velocity, can also be shown. The simulations created can be used in physics class to provide students with an interactive environment in which they can study Kepler’s laws and gravitational slingshot guided by the teacher to come to conclusions about the motions of the planets around the Sun and gravitational slingshot.

Keywords: Python, VPython, Kepler's laws, gravitational slingshot, computer simulation

Sadržaj

1 Uvod 1
2 Python 2
2.1 Razvoj Python-a 2
2.2 Sintaksa 3
2.3 Korištenje i rasprostranjenost Python-a 8
2.4 VPython 9
3 Simulacija Keplerovih zakona i gravitacijske praćke 12
3.1 Keplerovi zakoni 12
3.2 Gravitacijska praćka 14
3.1 Simulacije 15
3.2 Primjeri uporabe u nastavi 19
4 Zaključak 23
Dodaci 24
A Programska izvedba simulacije 24
Literatura 33

1. Uvod

Danas je sve veći fokus na istraživačkoj nastavi fizike koja rezultira boljim konceptualnim razumijevanjem fizike od klasične nastave [1]. Ključni dio istraživačke nastave su učenički pokusi i istraživanja [2]. Na nastavi možemo izvesti mnoge pokuse te demonstrirati razne pojave, no neke pojave predugo ili prekratko traju, njihovo promatranje zahtijeva sofisticiranu opremu ili su preopasne za promatranje u učionici. U takvim situacijama dobra zamjena su simulacije. One nam daju interaktivnu okolinu u kojoj učenici mogu samostalno ili u grupama proučavati i istraživati pojavu. Također pružaju učeniku manje stresnu okolinu za istraživanje neke pojave. Osim što se mogu koristit za prikaz nekih pojava koje iz raznih razloga ne možemo demonstrirati na nastavi, mogu nam poslužiti kao dopuna u proučavanju drugih, gdje u simulaciji možemo vizualizirati i neke dijelove pojava koji možda nisu lako uočljivi ili ih prikazati tako da budu lakše shvatljivi. Jedna od tema koju možemo obraditi uz pomoć simulacija su Keplerovi zakoni. Ime su dobili po njemačkom astronomu Johannes Kepleru. Oni opisuju pravilnosti u gibanjima planeta oko Sunca koje je u 17. stoljeću uočio Kepler [3]. Promatranje planetarnog gibanja bi dugo trajalo i zahtijevalo opsežna mjerenja, a simulacija koja ih vjerno prikazuje traje svega nekoliko minuta te učenici u kratkom roku mogu na temelju viđenog donositi vlastite zaključke. Osim Keplerovih zakona možemo simulirati i gravitacijsku praćku. Gravitacijska praćka je manevar u kojem međuplanetarne sonde za promjenu brzine koriste relativna gibanja planete i gravitacijsku silu te može biti zanimljiv primjer primjene fizikalnih zakona pri rješavanju inženjerskih problema. mehanizam gravitacijske praćke bio ključan za slanje letjelica Voyager 1 i 2 izvan granica Sunčevog sustava krajem 70-tih godina. Za izradu simulacija možemo koristiti razne programske jezike i pakete. Programski jezik Python je jednostavan i široko rasprostranjen te velik broj nastavnika i učenika već ima iskustva rada s njim. Također se koristi u nastavi informatike pa su učenici već s njime upoznati, što znači da osim samih fizikalnih pojava koje istražuje simulacija učenici mogu proučavati i kod simulacije, te se ovime ujedno uvodi interdisciplinarni pristup nastavi. Python nudi razne pakete za razne svrhe, jedan njegov paket je VPython koji značajno olakšava izradu 3d prikaza i simulacija. Koristeći VPython u svega par linija koda možemo izraditi jednostavne 3d prikaze kao što je gibanje kugle u 3d prostoru, ali također možemo izraditi i kompleksnije sustave. Koristeći Python s paketom VPython možemo napraviti razne simulacije koje će vjerno opisati neki stvarni

sustav te ga tako možemo proučavati, doći do zaključaka kako bi se ponašao stvarni sustav i istraživati kako promjene određenih početnih parametara utječu na ponašanje sustava.

2. Python

Python je dinamičko pisani, interpretirani jezik visoke razine s automatskim upravljanjem memorijom. Podržava više paradigmi programiranja: objektno orijentirano, proceduralno i funkcionalno programiranje. Dostupan je za mnoga radna okruženja kao što su Windows, Mac-OS, Linux [4].

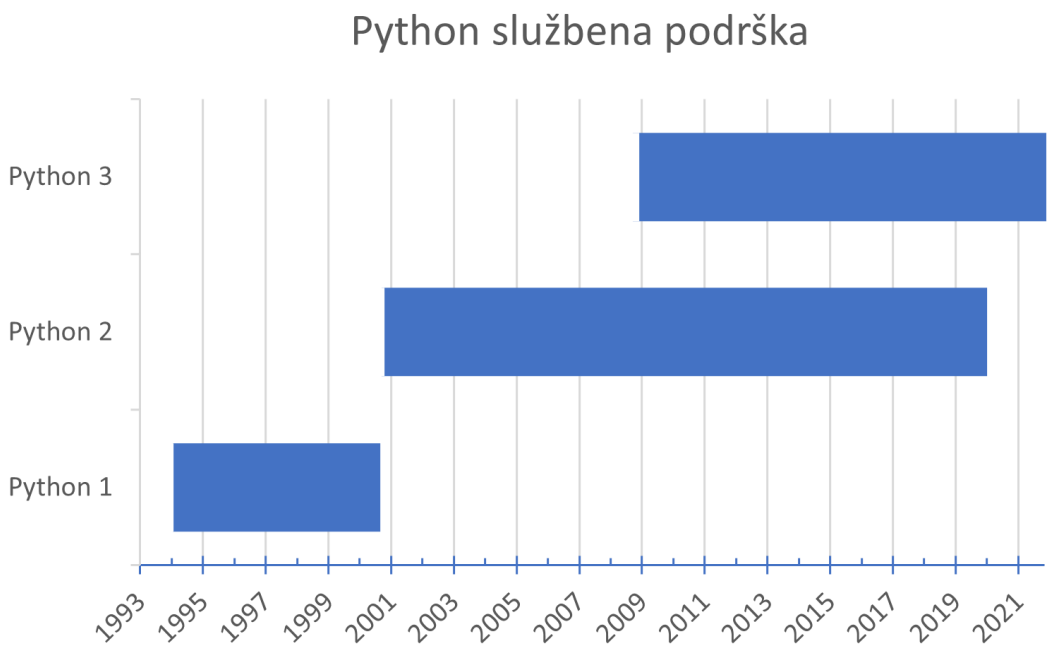
Python koristi uvlačenja za grupiranje iskaza. Ta značajka čini kod čitljivijim na dva načina. Prvo, korištenje uvlačenja smanjuje vizualni nered i skraćuje programe, čime se smanjuje raspon pozornosti potreban za uzimanje osnovne jedinice koda. Drugo, omogućuje programeru manje slobode u oblikovanju, čime se omogućuje ujednačeniji stil, što olakšava čitanje tuđeg koda [5]. Python je dinamički pisani programski jezik te nije potrebno definirati tip varijable i one mogu mijenjati tip kroz izvođenje programa [6]. Također ima automatsko upravljanje memorijom: „Sakupljač smeća“. On upravlja rezerviranjem i oslobađanjem memorije za potrebe aplikacije [7]. To znači da se ne mora pisati kod za upravljanje memorijom, te tako se smanjuje mogućnost za „curenjem memorije“ tj. da se zaboravi osloboditi rezervirana memorija kad se prestane koristiti.

2.1 Razvoj Python-a

Razvoj je počeo u prosincu 1989., kao "hobi" programski projekt Guido van Rossuma. Zamišljen je kao nasljednik programskog jezika ABC, namijenjenog neprofesionalnim programerima. Korištenje uvlačenja za grupiranje izraza proizlazi upravo iz ABC-a. Python ime dobiva po televizijskoj seriji "Monty Python's Flying Circus", koje je osnivač Guido van Rossum bio veliki obožavatelj [5]. U veljači 1991. Van Rossum je objavio verziju 0.9.0 na alt.sources. U ovoj fazi razvoja već su bile prisutne klase s nasljeđivanjem, rukovanje iznimkama, funkcijama i osnovnim tipovima podataka. Verzija 1.0 je izdana u siječnju 1994. koja među ostalim dodaje funkcije map, filter, reduce i novu ključna riječ lambda. Sljedeća glavna verzija je Python 2.0, objavljen u listopadu 2000. Verzija 2.0 dodaje jednu od najtraženijih značajki, 11 novih operatora dodjeljivanja: += ,-= , *=, /=, %=, **=, <<=, >>=, &=, ^=, |=, te dodaje neke nove funkcije i veću kontrola nad sakupljanjem smeč [8]. Podrška za Python 2.x, točnije 2.7 završila je 1. siječnja 2020 [9]. Posljednje izdanje, 2.7.18, objavljeno je 19. travnja 2020 [10]. Trenutno je zadnja glavna verzija Python 3.0, izdana je u prosincu 2008. godine.

Zadnja verzija je 3.10.4 izdana u ožujku 2022 [11]. Python 3.0 narušio je kompatibilnost unatrag, a veći dio koda za Python 2 ne radi nepromijenjen na Python-u 3. napuštanje pune kompatibilnosti omogućilo je veće, potrebne promjene koje se nisu mogle implementirati uz zadržavanje pune kompatibilnosti unatrag s 2.x serijom. Neke od promjena koje donosi su:

- Promjena ispisa tako da bude ugrađena funkcija, a ne izjava. To je olakšalo promjenu modul da koristi drugu funkciju ispisa, kao i pravilniju sintaksu.
- Promjena cjelobrojnog dijeljenja, koje je u Pythonu 2 uvijek vraćalo cijeli broj. Npr. 5/2 je davalo 2, a u Python-u 3 je 5/2 = 2.5 , dodaje se novi operator za cjelobrojno dijeljenje (5//2=2)
- Značajne promjene u binarnim i Unicodeu podacima [12].



Graf 2.1 Prikaz službene podrške za verzije Pythona po godinama.

2.2 Sintaksa

2.2.1 Varijable

Varijabla je naziv za određenu vrijednost spremljenu u memoriji. U Python-u nije potrebno prethodno definirati varijablu prije dodjeljivanja vrijednost. Postavljanje vrijednosti se vrši pomoću operatora dodjeljivanja (=).

Python ima mogućnost pisanja komentara, napisani dio teksta koji se ne izvodi, a označava se s znakom „#“ i ide do kraja tog reda.

```
my_var = 21        # varijabla imena my_var ima vrijednost 21
my_var = "abc"     # varijabla imena my_var ima vrijednost abc
```

Slika 2.1. Primjer sintakse za dodjeljivanje vrijednosti.

Imena varijabli ne mogu sadržavati posebne znakove, počinjati brojem ili biti jedna od ključnih riječi [13].

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Tablica 2.1. Ključne riječi u Python-u.

2.2.2 Operatori

Operator prihvaća jedan ili više ulaza u obliku varijabli ili izraza, obavlja zadatak (kao što je usporedba ili zbrajanje), a zatim daje izlaz u skladu s tim zadatkom. U Python-u razlikujemo nekoliko tipova operatora:

Aritmetički (+, -, *, /, %, **, //)

Relacijski (==, !=. <, >, >=, <=)

Logički (and, or, not)

Bitni (&, |, ^, ~, <<, >>)

Operatori dodjele (=, +=, -=, *=, %=, //=, **=, &=, |=, ^=, >>=, <<=)

Operater članstva (in, not in)

Operatori identiteta (is, is not)

2.2.3 Funkcije

Funkcije su objekti stvoreni definicijama funkcija. Jedina operacija na funkcijskom objektu je pozivanje. Tijekom pozivanja funkcije prosljeđujemo joj argumente [14].

```
def my_function(x):
  return 5 * x

print(my_function(3))   #Pozivamo funkciju s argumentom 3
                        #funkcija vraća 15 te program ispisuje 15
```

Slika 2.2 primjer sintakse funkcije.

2.2.4 Greške i iznimke

Postoje dvije različite vrste pogrešaka: *sintaktičke pogreške* i *iznimke*. *Sintaktičke pogreške* govore da izraz sintaktički nije točan, a *iznimke* da izraz uzrokuje pogrešku kada se pokuša izvršiti [15].

2.2.5 Tipovi podataka

Numerički tipovi podataka su cijeli brojevi, brojevi s pomičnim zarezom i kompleksni brojevi te Boolen, oni su podvrsta cijelih brojeva. Mješovita aritmetika je u potpunosti podržana. Tijekom operacija operand s "užim" tipom se proširuje, dakle cijeli broj se proširuje u broj s pomičnim zarezom, a on u kompleksni. U Python-u možemo raditi s brojevima u različitim bazama [14]. Naprimjer heksadecimalne brojeve označava prefiks 0x a binarne 0b.

```
X = 0xff        #x ima vrijednost 255
Y = hex(255)    #y ima vrijednos '0xff'
```

Slika 2.3. Primjer rada s heksadecimalnim brojevima.

Nizovi su skup podataka koji su uređeni niz te dozvoljavaju duplikate. Neki od nizova su: *liste* i *n-torke* [14].

N-torke su nepromjenjivi tip i jednom kad ih definiramo ne možemo ih mijenjati.

```
t = ()            #prazna n-torka
t = tuple(a,b,c)
t = (a,b,c)
t = a,b,c
```

Slika 2.4 Primjer sintakse za definiranje *N-torke.*

Liste su promjenjivi tip i njih možemo mijenjati.

```
l=[]
l=list()
l=[a,b,c]
```

Slika 2.5 Primjer sintakse za definiranje *liste.*

Tekstualni niz (string) se koristi u Python-u za rukovanje tekstualnim podatcima. To je *niz* unicode elemenata te implementiraju sve uobičajene operacije *nizova*, zajedno s dodatnim metodama [14].

```
str = 'allows embedded "double" quotes'
str = "allows embedded 'single' quotes"
str = '''Three single quotes'''
str = """Three double quotes"""
```

Slika 2.6 Primjer sintakse za definiranje *string* objekta.

Set je neuređena zbirka različitih objekata. Uobičajene upotrebe uključuju testiranje članstva, uklanjanje duplikata iz *niza* i računanje matematičkih operacija kao što su presjek, unija, razlika i simetrična razlika. Budući da je set neuređen, ne bilježi položaj elementa ili redoslijed umetanja. Sukladno tome, skupovi ne podržavaju indeksiranje, rezanje ili drugo ponašanje slično nizovima [14].

```
s = {'Zemlja','Mars'}
s = set(['a', 'b', 'c'])
```

Slika 2.7 Primjer sintakse za definiranje *seta.*

Rječnici su ključ-vrijednost parovi te svaki ključ ima pridruženu vrijednost kojoj možemo pristupiti koristeći taj ključ [14].

```
d = {}
d = { 'foo':123 , 'bar':200}
d = dict(foo=100, bar=200)
```

Slika 2.8 Primjer sintakse za definiranje *rječnika.*

2.2.6 Kontrola toka

Za kontrolu toka koristimo grananja i petlje. Jedna naredba grananja je *if-else* izjava. Ako je izraz *if* izjave istinit onda se izvršava njen kod, ako nije onda se izvodi kod unutar izjave *else*. Ukoliko želimo provjeriti više uvjeta prije izvođenja koda unutar else možemo koristiti *elif* nakon *if* izraza [16].

```
if x<0:
   print("x je negativan") #Ako je x manje od 0 ispisuje x je negativan.
 elif x==0:
   print("x je nula")      #Ako je x 0 ispisuje x je 0.
 else:
   print("x je pozitivan") # Ako x nije manji od nule i nije jednak nuli
                           #ispisuje x je pozitivan.
```

Slika 2.9 Primjer sintakse *if* izjave.

For izjava je petlja koja iterira kroz elemente bilo kojeg niza, redoslijedom kojim se pojavljuju u nizu [16].

```
rijeci = ['zvijezda', 'planet', 'Mars']
for r in rijeci:
  print(r)              #Petlja iterira kroz elemente te ih ispisuje
```

Slika 2.10 Primjer sintakse *for* petlje.

Ako trebamo iterirati niz brojeva, koristimo funkciju *range( )* [16].

```
for i in range(5):
  print(i)              #Petlja ispisuje brojeve 0,1,2,3,4
```

Slika 2.11 Primjer sintakse *for* petlje.

Druga petlja koju imamo je izjava *while,* ona ponavlja kod unutar izjave dok god je izraz istinit.

```
a = 0
while a < 10:           #Petlja se ponavlja dok je a manje od 10
  print(a)              #Ispisuje a
  a = a + 1             #Pvećava a za jedan
                        #Nakon izvođenja će biti ispisani brojeve 0-9
```

Slika 2.12 Primjer sintakse *while* petlje.

Izjave *break* i *continue* možemo koristiti unutra petlji [16]. Naredba *break* nam služi da izađemo iz petlje.

```
for i in range(5):
  if i==3:              #Ako je i jednako 3
    break               #izlazi iz petlje.
  print(i)              #Biti će ispisani brojevi od 0 do 3.
```

Slika 2.13 Primjer sintakse *for* petlje s *break* izjavom.

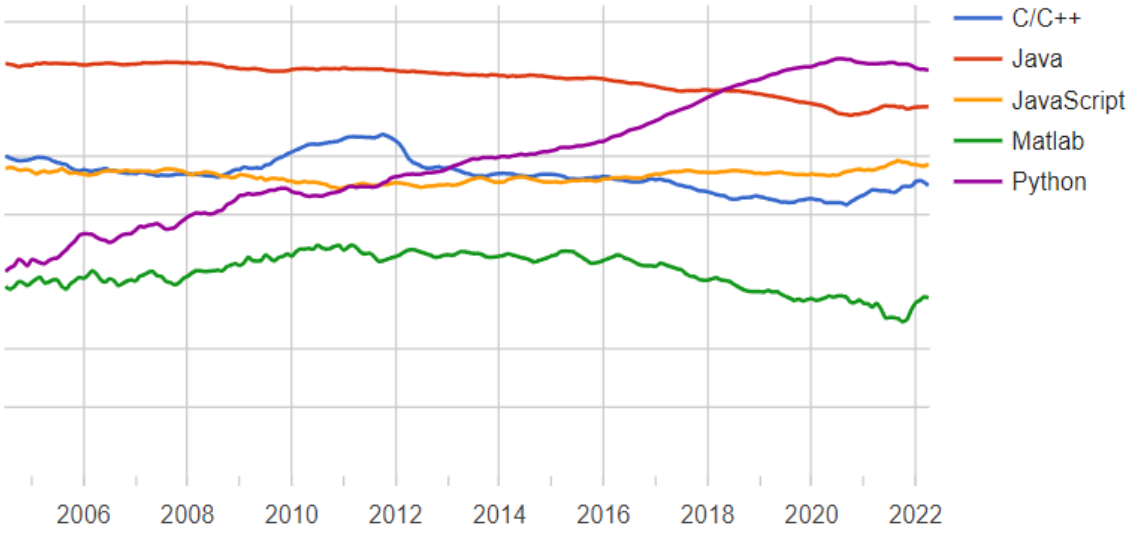
Naredba *continue* preskače dio koda nakon naredbe i počinje novu iteraciju petlje.

```
for i in range(5):
 if i==3:               #Ako je i jednako 3
   continue             # preskače tu iteraciju
 print(i)               #Ispisati će brojevi od 0,1,2,4,5. broj 3 neće
                        biti ispisan jer je print u toj iteraciji
                        preskočen
```

Slika 2.14 Primjer sintakse *for* petlje s *continue* izjavom.

2.3 Korištenje i rasprostranjenost Python-a

Python je danas jedan od najpopularnijih jezika, koristi se u raznim područjima kao što su razvoj programske potpore za web, znanost, analiza podataka, numeričke metode, strojno učenje, razvoj korisničkih sučelja za stolna računala, razvoj softvera [17]. Python je trenutno najpopularniji jezik po PYPL Indeksu koji je baziran na podatcima iz Google Trends-a te pretpostavlja što su više pretraživani edukacijski materijali za jezik to je jezik popularniji [18].



Graf 2.2 Popularnost Python-a po PYPL indexu.

Zbog velikog broja modula uključenih u Python distribuciju kažemo da dolazi s „uključenim baterijama“. No ako standardna distribucija ne uključuje ono što je potrebno Python Package Index (PyPI), sadrži preko 370000 paketa sa širokim rasponom funkcionalnosti [19]. Neki od popularnijih paketa korištenih u znanosti i obrazovanju su:

- NumPy nudi visoko optimizirane opsežne matematičke funkcije, generatore slučajnih brojeva, rutine linearne algebre, Fourierove transformacije, alate za integraciju C/C++ koda, i još mnogo toga [20].
- *SciPy* Proširuje *NumPy* pružajući dodatne alate za računanje nizova i pruža specijalizirane strukture podataka, kao što su rijetke matrice i k-dimenzionalna stabla. Nudi algoritme za rješavanje algebarskih jednadžbi, diferencijalnih jednadžbi i mnoge druge probleme [21].
- *SymPy* je Python biblioteka za simboličku matematiku. Cilj mu je postati potpuno opremljeni sustav računalne algebre (CAS) uz istovremeno održavanje koda što

jednostavnijeg kako bi bio razumljiv i lako proširiv. *SymPy* je u potpunosti napisan na Pythonu i ne zahtijeva nikakve vanjske biblioteke [22].

- *Pandas* je brz, moćan, fleksibilan i jednostavan za korištenje alat za analizu i manipulaciju podataka [23]. Nudi mnoge alate za rad s tabličnim podatcima i obradu podataka.
- *Matplotlib* je 2D biblioteka za crtanje. Sa samo nekoliko linija koda može generirati dijagrame, histograme, grafikone grešaka, dijagrame raspršenja, itd., sa samo nekoliko redaka koda. *Matplotlib* je opsežna biblioteka za stvaranje statičnih, animiranih i interaktivnih vizualizacija u Pythonu [24].
- *Jupyter Notebook* je interaktivna računalna platforma temeljena na webu. „Bilježnica“ kombinira živi kod, jednadžbe, narativni tekst, vizualizacije i druge medije.
- *VPython* olakšava stvaranje 3D prikaza i animacija [25].

Python se u Republici Hrvatskoj koristi na nastavi informatike kao jezik u kojem se uči programiranje [26]. Python sve više prodire u sve razine obrazovanja. Nudi okruženje u kojem se mogu istražiti razni tipovi programiranja kao što su objektno orijentirano, proceduralno i funkcionalno programiranje. Njegova jednostavnost i jasna sintaksa, automatsko upravljanje memorijom i činjenica da je dinamički pisan čine ga idealnim prvim jezikom. Veliki broj postojećih knjižnica čini ga prikladnim za lako rješavanje gotovo svih jednostavnijih ili kompliciranijih programskih zadataka i problema.

2.4 VPython

VPython olakšava stvaranje 3D prikaza i animacija, čak i za korisnike s ograničenim iskustvom u programiranju. Budući da se temelji na Pythonu, također ima mnogo toga za ponuditi iskusnim programerima i istraživačima [27].

2.4.1 3D prikazi

Koristeći VPython u nekoliko linija koda možemo stvoriti grafičke prikaze.

```
ball = sphere(pos=vector(2,2,2),radius=0.7)
#Stvara sferu sa središtem na lokaciji (2,2,2) s radijusom = 0,7.
```

Slika 2.15. Primjer koda za prikaz 3D kugle.

Stvaranjem 3D objekata oni se automatski dodaju u zadnje definirani prikaz. U početku se automatski stvara 3D VPython *canvas* prikaz naziva *scene* [28].

```
scene = canvas()          # automatsko te nije potrebno pisati
box(...)                  # pojaviti ce se na "scene"
sphere(canvas=scene, ...) # pojaviti ce se na "scene"

scene2 = canvas( )
box(...)                # pojaviti ce se na "scene2"
sphere(canvas= scene2, ...)  # pojaviti ce se na " scene2"
cone(canvas=scene, ...) # pojaviti ce se na "scene"
```

Slika 2.16. Primjer koda za dodavanje objekata na canvas.

2.4.1 2D prikazi

Osim 3D *canvas* možemo definirati i graf. Za razliku od *canvas* koje je inherentno 3D i sadrži 3D objekte kao što su kugle i kutije, graf je inherentno 2D i sadrži grafičke objekte za prikaz krivulja, točaka i okomitih ili horizontalnih stupaca [29].

```
f = gcurve(color=color.red)   # Graficka linija
for x in range(0, 5, 0.1):    # x ide od 0 do 5 u intervlima od 0.1
    f.plot( x, cos(x) )
```

Slika 2.17. Primjer koda za ispis kosinus krivulje.

Povezana krivulja (gcurve) je jedna od nekoliko vrsta objekata za crtanje grafova. Druge opcije su nepovezane točke (gdots), okomiti stupci (gvbars) i horizontalne stupci (ghbars).

```
g1 = graph()          # definiramo graf
gcurve(...)           # krivulja će se pojaviti na grafu "g1"
gdots(graph=g1, ...) # krivulja će se pojaviti na grafu "g1"

g2 = graph()
gcurve(...)            # krivulja će se pojaviti na grafu "g2"
gdots(graph=g2, ...)  # točke će se pojaviti na grafu "g2"
gvbars(graph=g1, ...) # stupci će se pojaviti na grafu "g1"
```

Slika 2.18. Primjer koda za dodavanje objekata na graf.

2.4.3 Animacija

Nakon stvaranja objekta lako možemo mijenjati njegova svojstva:

```
ball.pos = vector(0,0,0)    # postavi u ishodište
ball.pos.x = 11             # promjeni pos.x
ball.color = vector(0,1,0)  # postavi boju na zelenu
```

Slika 2.19. Primjer koda za promjenu svojstva objekta.

Animacije stavljamo unutar petlje

```
while 1:
  rate(50)                    #maximalan broj poziva u 1s je 50
  ball.pos.x = ball.pos.x + 1 #povecaj poziciju na x za 1
```

Slika 2.20. Primjer koda za animaciju pomicanja objekta u x smjeru.

Da bi bilo moguće ažuriranje zaslona ili obrada pritiska tipki tipkovnice ili miša u petlju je potrebno staviti funkciju *rate(f)*. Funkcija *rate(frekvencija)* zaustavlja izračune do 1,0/frekvencija sekunde nakon prethodnog poziva *rate()* te se petlja izvodi maksimalno *frekvencija* broj puta u sekundi. Na primjer, rate(50) će zaustaviti izračune dovoljno dugo da prođe najmanje 1,0/50,0 = 0,02 sekunde. Druga opcija je korištenje funkcije spavanja*:* *sleep(0,02)* znači "ne radi ništa 0,02 sekunde" i ekvivalentno je *rate(50)* [30]. Funkcija *rate()* stvara pauzu u izvođenju petlje te daje programu vremena da obradi događaje miša ili tipkovnice i ažurira zaslon s novim stanjem animacije.

3. Simulacija Keplerovih zakona i gravitacijske praćke

3.1 Keplerovi zakoni

Ljudi su promatrali kretanje planeta, zvijezda i drugih nebeskih tijela tisućama godina. Zapravo mnogi korisni matematički koncepti, kao što su Besselove funkcije i nelinearni najmanji kvadrati, proizlaze iz tih promatranja [31]. Za Astronomiju se može reći da potječe iz Babilona gdje su imali izvanredno točno znanje o periodima Sunca, Mjeseca i planeta te su mogli predvidjeti položaj tih tijela među zvijezdama i ponavljanje pomrčina Mjeseca. Grčki znanstvenici su proučavanjem kretanja nebeskih tijela došli do Geocentričnog modela svemira u kojem je Zemlja u središtu svemira. Grčki astronom Klaudije Ptolemej u drugom stoljeću je formalizirao model u kojemu je nebo sfera koja rotira oko fiksne osi. Zemlja je sfera koja miruje u centru neba. Zemlja, u usporedbi s nebom, je puno manja, ona je tek točka. Planeti se gibaju manjim kružnicama nazvanim epicikl čiji centar kruži većom kružnicom nazvanom deferent u čijem je centru zemlja. Taj model je bio prihvaćen sljedećih 1400 godina. Početkom 16. stoljeća poljski astronom Nikola Kopernik je primijetio nelogičnosti u geocentričnom sustavu te je zaključio da bi sustav u kojem Zemlja ne miruje riješio te nelogičnosti. Sugerirao je Heliocentrični model u kojem Zemlja nije u centru već se Zemlja i drugi planeti gibaju u kružnim orbitama oko Sunca. Krajem 16. stoljeća danski astronom Tycho Brahe izveo brojna precizna promatranja nebeskih tijela. Ta promatranja planeta i 777 zvijezda vidljivih golim okom izvedeni su samo velikim sekstantom i kompasom. On se nije slagao s Heliocentričnim modelom Kopernika, točnije s idejom Zemlje koja se kreće te je predlagao drugi model u kojemu Sunce i Mjesec kruže oko Zemlje, a ostali planeti kruže oko Sunca. Njemački astronom Johannes Kepler kratko vrijeme je bio Braheov pomoćnik prije Braheove smrti, nakon čega je dobio astronomske podatke svog mentora i proveo 16 godina pokušavajući zaključiti matematički model za kretanje planeta. Nakon mnogih napornih proračuna Kepler je iz podataka o revoluciji Marsa oko Sunca došao do uspješnog modela [3]. U 17. stoljeću Kepler je unaprijedio heliocentričnu teoriju Nikole Kopernika modelom u kojemu se planeti oko Sunca gibaju po eliptičnim putanjama sa Suncem u fokusu. Opisuje brže gibanje planeta kada su bliže Suncu te daje precizan odnos veličina planetarnih orbita i vremena potrebnog da planet obiđe Sunce. Iako je opisao gibanje planeta, nije definirao uzrok tog gibanja niti je dao matematički opis. Tek nekoliko desetljeća kasnije, Newton je razvio matematički model koji je podržavao Keplerove dedukcije.

Keplerova potpuna analiza gibanja planeta može se sažeti u tri tvrdnje poznate kao Keplerovi zakoni:

1. *Planeti se gibaju po elipsama oko Sunca, koje se nalazu u jednom od žarišta (fokusa) elipse [32].*

   Ovaj zakon slijedi iz Newtonovog zakona gravitacije pod uvjetom da je masa središnjeg tijela znatno veća od mase planeta i da se učinak planeta na središnje tijelo može zanemariti. Tada se planeti gibaju po putanji:

   $$\mathrm{r}(\theta) = \frac{p}{1 + \varepsilon\, cos(\theta)} \tag{3.1}$$

   točnije po elipsi, gdje je p parametar orbite dok je ε ekscentritet. Kada je $0 < \varepsilon < 1$ onda je orbita elipsa, dok u graničnom slučaju $\varepsilon = 0$, orbita je kružnica sa središnjim tijelom u središtu [33].

2. *Radijus vektor r u jednakim vremenskim intervalima prebrisuje jednake površine P[32].* Ovaj zakon proizlazi iz očuvanja kutnog momenta. Površina koju radijus vektor prebriše tijekom kratkog vremenskog razdoblja je dana izrazom (3.2) gdje je $r\, d\theta$ baza tankog trokuta koji prebriše vektor r u intervalu dt:

   $$dA = r(rd\theta) \,/\, 2 \tag{3.2}$$

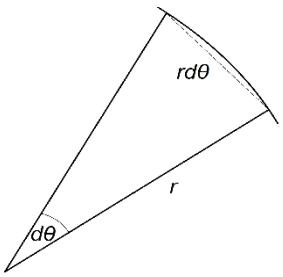
   koristeći:

   $$L = mr^2\dot{\theta} \tag{3.3}$$

   Gdje je L moment količine gibanja* dobijemo:

   $$\frac{dA}{dt} = \frac{1}{2}r^2\theta = \frac{\dot{L}}{2m} \tag{3.4}$$

   što je konstantna, budući da je L konstantna za centralnu silu [33].



Slika 3.1. Prikaz tankog trokuta koji prebriše vektor r u intervalu dt.

* Putanje u polju centralne sile nalaze se u ravnini koja je okomita na moment količine gibanja pa je stoga L ukupni moment

3. *Kvadrati ophodnih vremena T oko Sunca odnose se kao kubovi srednjih udaljenosti r od Sunca* [32].

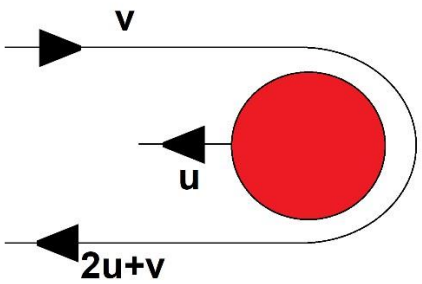
   Točnije:

$$T^2 = \frac{4\pi^2 m a^3}{\alpha} \equiv \frac{4\pi^2 a^3}{GM} \tag{3.5}$$

Gdje je M masa Sunca. Vidimo da je omjer a^3/T^2 konstanta [33].

3.2 Gravitacijska praćka

Gravitacijska praćka je manevar koji koristi relativno kretanje kao što je gibanje planeta orbitama oko Sunca i gravitaciju planeta ili drugog astronomskog objekta za promjenu brzine letjelice radi uštede goriva i smanjenja troškova.



Slika 3.2. Prikaz manevra gravitacijske praćke.

Gravitacije se može koristiti za ubrzanje letjelice, odnosno za povećanje ili smanjenje njezine brzine ili preusmjeravanje njezine putanje. "Pomoć" pruža kretanje tijela dok ono povlači letjelicu. Svaki dobitak ili gubitak kinetičke energije i brzine svemirske letjelice u prolazu se na odgovarajući način gubi ili dobiva od strane gravitacijskog tijela, u skladu s trećim Newtonovim zakonom. No zbog puno veće mase u odnosu na masu letjelice ta promjena energije je zanemariva. Gravitacijska praćka prvi je put korištena 1959. godine kada je sovjetska sonda Luna 3 fotografirala dalju stranu Zemljinog Mjeseca, a danas je koriste gotovo sve interplanetarne sonde [34]. Do izraza možemo lako doći. Količina gibanja i kinetička energija prije i poslije sudara su očuvani. Neka su u_1 i u_2 brzine prije sudara a v_1 i v_2 brzine nakon sudara Količina gibanja i kinetička energija prije i nakon sudara je:

$$m_1 u_1 + m_2 u_2 = m_1 v_1 + m_2 v_2 \tag{3.6}$$

$$\frac{1}{2}m_1u_1^2 + \frac{1}{2}m_2u_2^2 = \frac{1}{2}m_1v_1^2 + \frac{1}{2}m_2v_2^2 \tag{3.7}$$

Sada rješavanjem tih jednadžbi dodamo do izraza za brzine nakon sudara:

$$v_1 = \frac{m_1 - m_2}{m_1 + m_2}u_1 + \frac{2m_2}{m_1 + m_2}u_2 \tag{3.8}$$

$$v_2 = \frac{2m_1}{m_1 + m_2}u_1 + \frac{m_2 - m_1}{m_1 + m_2}u_2 \tag{3.9}$$

U slučaju letjelice i planeta masa planeta je puno veća od mase letjelice pa je masa letjelice u odnosu na masu planeta zanemariva te jedna a i b postaju:

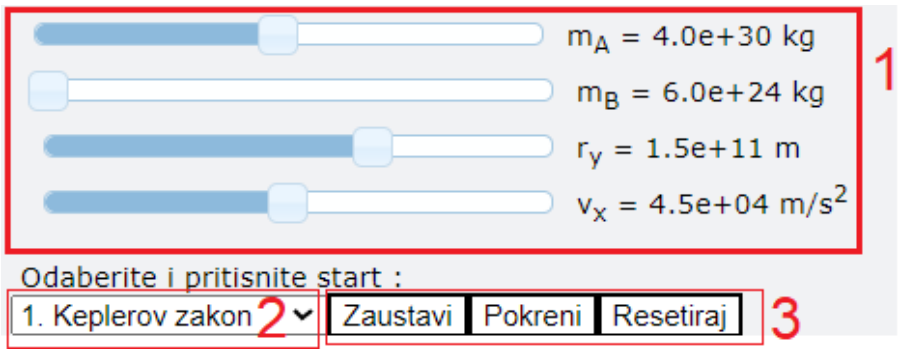
$$v_1 \approx -u_1 + 2u_2 \tag{3.10}$$

$$v_2 \approx u_2 \tag{3.11}$$

3.3 Simulacije

Simulacije su napravljene korištenjem vPython 7 paketa za stvaranje 3d animacija. Simulacije prikazuju 2 tijela koja međudjeluju gravitacijskom silom, te prikazuje Keplerove zakone i gravitacijsku praćku.

Prilikom korištenja u korisničkom sučelju u području 1 odabiru se početni parametri: Masa tijela A m_a, masa tijela B m_b, početna udaljenost tijela B r_y te početna brzina tijela B v_x. Pritiskom na padajući izbornik u području 2 se odabire scenarij simulacije. Ponuđeni scenariji su: 1. Keplerov zakon, 2. Keplerov zakon, 3. Keplerov zakon te gravitacijska praćka. Nakon odabira upravljamo tijekom simulacije korištenjem tipki u području 3. Pritiskom tipke start se započinje simulacija, tipka stop zaustavlja simulaciju a tipka reset vraća parametre na početne te čisti sve ispisano na 3d grafu i prikazu.



Slika 3.3. Prikaz korisničkog sučelja simulacije.

Simulaciju možemo pokretati s različitim početnim uvjetima te promatranjem rezultata možemo donijeti razne zaključke. Na simulacijama vidimo 3d prikaz gibanja te prikaz na grafu na kojem se ispisuju relevantni podatci vezani uz svaku simulaciju. Za izračune pozicije u svakom vremenskom trenutku simulacija se koriste Newtonovi zakoni gibanja i Newtonov zakon gravitacije. Iz Newtonovog zakona gravitacije dobijemo silu kojom jedno tijelo djeluje na drugo, G je gravitacijska konstanta, m_1 je masa prvog tijela, a m_2 je masa drugog tijela, $\vec{r}_{12}$ je vektor udaljenosti ta dva tijela dva tijela (3.12)

$$\vec{F} = -G\frac{m_1 m_2}{|r_{12}|^2}\hat{r}_{12} \tag{3.12}$$

Iz Newtonovog zakona znamo da sila kojom tijelo jedan djeluje na tijelo dva je iznosom ista sili kojom tijelo dva djeluje na tijelo jedan, dakle :

$$\vec{F}_{12} = -\vec{F}_{21} \tag{3.13}$$

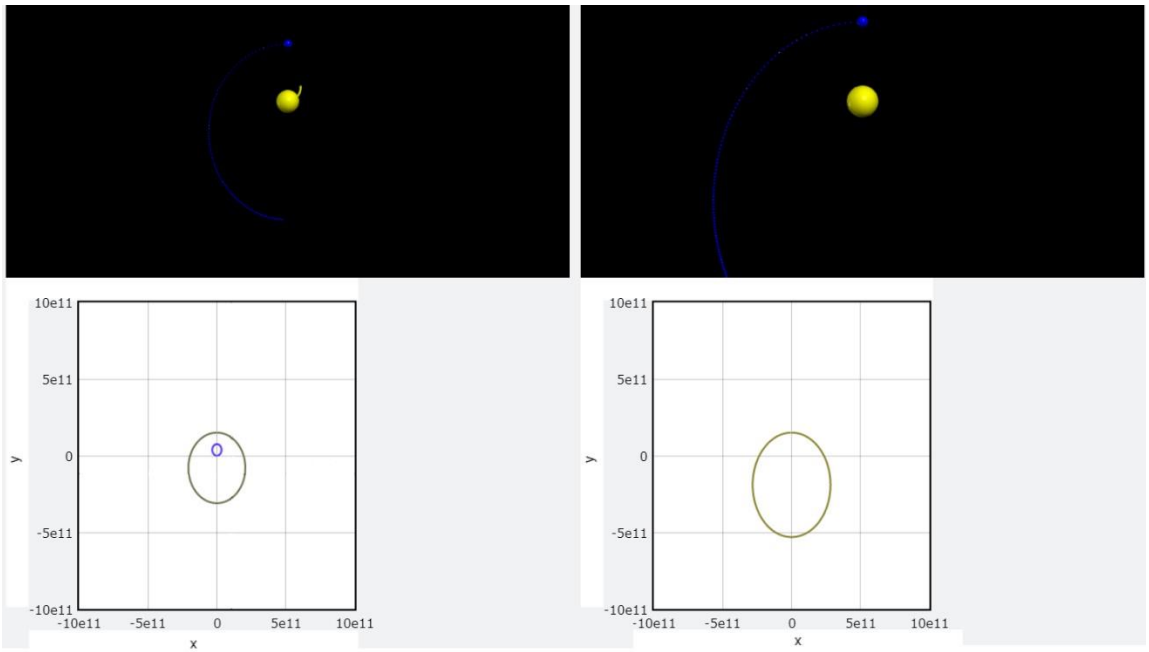
Kada znamo sile koje djeluju na tijelo onda možemo dobiti promjenu količine gibanja i količinu gibanja tijela. $\vec{P}_0$ je količina gibanja koju tijelo ima, $\vec{F}$ je sila koja djeluje na tijelo, onda količina gibanja $\vec{P}$ nakon kratkog vremenskog intervala dt tijekom kojeg sila $\vec{F}$ dijeluje na to tijelo je:

$$\vec{P} = \vec{P}_0 + \vec{F}dt \tag{3.14}$$

Iz količine gibanja znamo kojom se brzinom giba tijelo, a znajući brzinu možemo izračunati promjenu pozicije tijekom trenutka *dt* i novu poziciju

3.3.1 Simulacija 1. Keplerovog zakona.

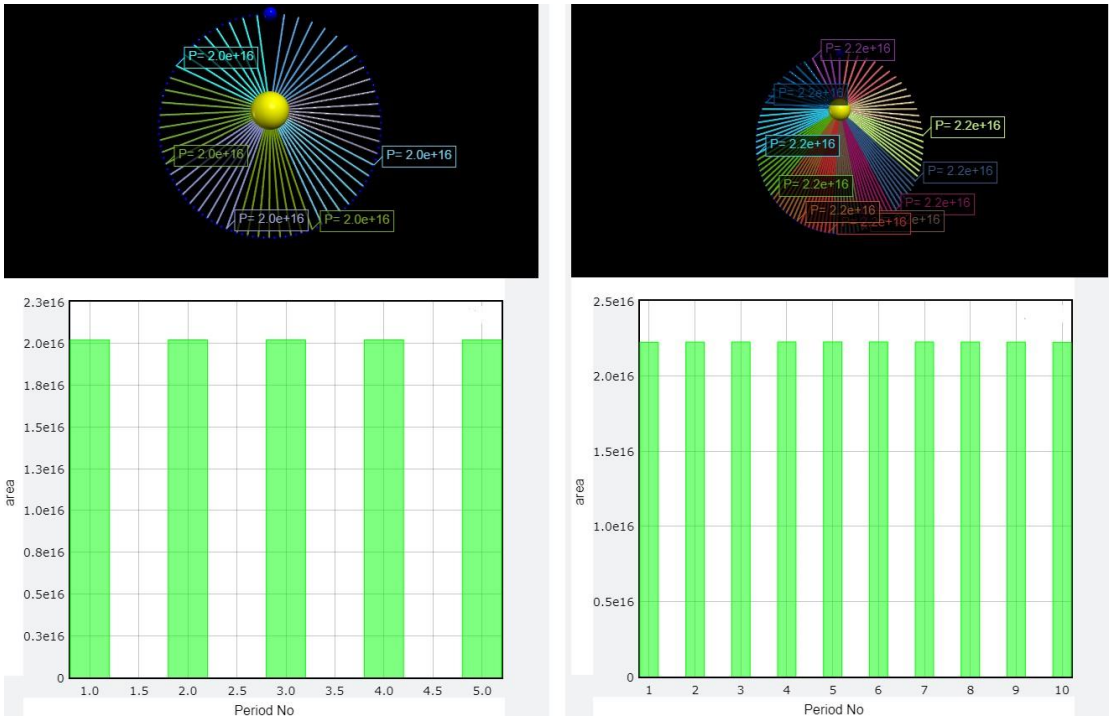
Odaberemo prvi Keplerov zakon i pokrenemo simulaciju. Simulacija prikazuje gibanje u 3d prostoru te iscrtava graf orbita tijela. Iz simulacije uočavamo da su putanje elipse te da promjenom početnih parametara one mijenjaju svoj oblik. Vidimo kada masa tijela B nije zanemariva u usporedbi s tijelom A (slučaj lijevo) tada se oba tijela gibaju u eliptičnim putanjama oko centra mase, a u slučaju kada tijelo a ima puno manju masu od tijela b tijelo A miruje u žarištu elipse kojom se giba tijelo B, što i očekujemo iz 1. Keplerovog zakona.



Slika 3.4. Primjer simulacije 1. Keplerovog zakona. Lijevo kada su mase približno iste, desno kada je masa planeta puno veća od mase Sunca.

3.3.2 Simulacija 2. Keplerovog zakona.

Da bi smo prikazali drugi Keplerov zakon u korisničkom sučelju odaberemo 2. Keplerov zakon i pokrenemo simulaciju, postavimo parametre te započnemo simulaciju.

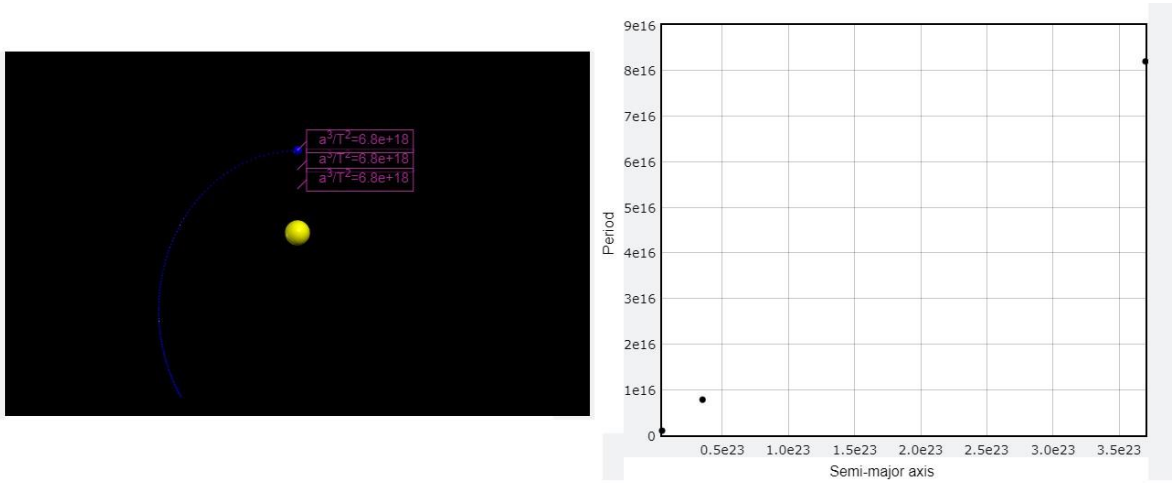


Slika 3.5. Primjer simulacije 2. Keplerovog zakona

Simulacija u 3d prikazu prikazuje gibanje te iscrtava površine P koje prebrisuje radijus vektor r u jednakim vremenskim intervalima i ispisuje njihove iznose. radijus vektor r je vektor koji spaja tijelo A i B. Na grafu se prikazuju iznosi tih površina te iz grafa lako možemo vidjeti da su one jednake. Iako se za različite putanje te površine mogu razlikovati, unutar iste putanje su one jednake. Simulacija ne računa točne površine već radi aproksimacije površinama trokuta koje zatvara radijus vektor u dva uzastopna trenutka u kojima simulacija računa pozicije.

3.3.3 Simulacija 3. Keplerovog zakona.

U korisničkom sučeljenu odaberemo 3. Keplerov zakon, postavimo parametre te započnemo simulaciju. Simulacija prikazuje gibanje, ispisuje omjer a^3/T^2 i te vrijednosti na grafu. Nakon nekoliko mjerenja lako uočimo da je omjer uvijek konstantan.

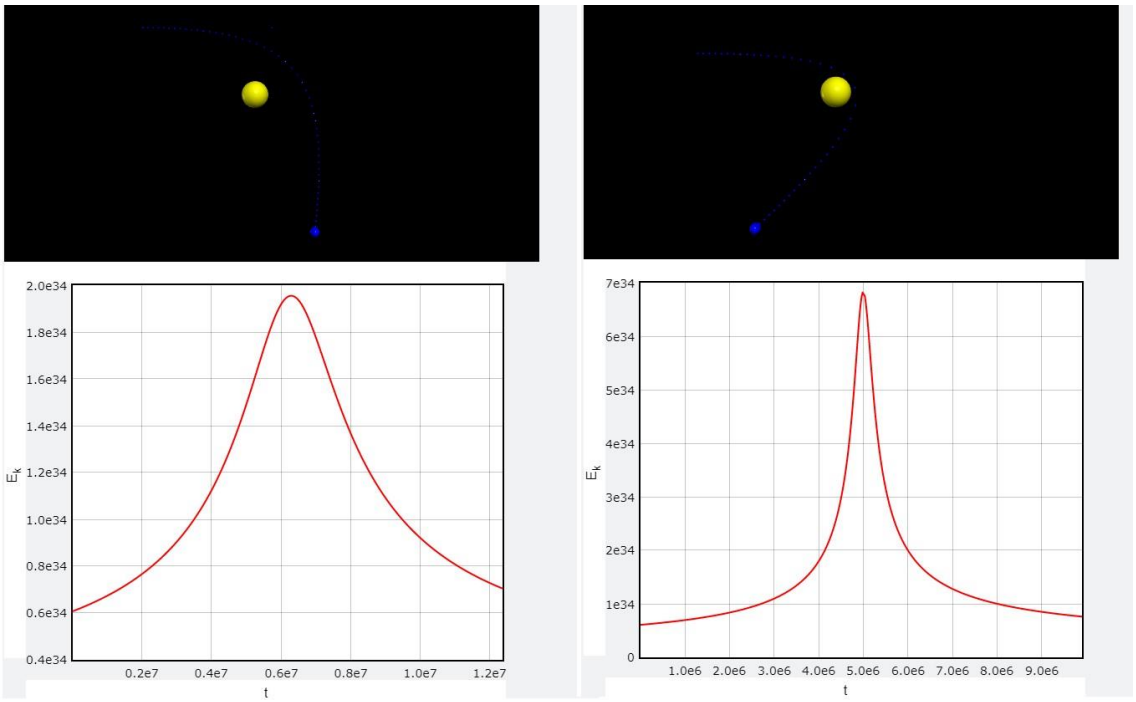


Slika 3.6. Primjer simulacije 3. keplerovog zakona.

Simulacija računa vrijednost glavne poluosi iz vrijednosti glavne osi koju pronalazi tako da mjeri i uspoređuje sve radijus vektore *r* kroz cijelu putanju te tako pronađe vrijednost glavne poluosi.

3.3.3 Simulacija gravitacijske praćke.

U korisničkom sučeljenu odaberemo gravitacijsku praćku te započnemo simulaciju.



Slika 3.7. Primjer simulacije gravitacijske praćke.

Odabirući različite parametre možemo vidjeti da dobijemo razne putanje. Iz 3d prikaza također vidimo da se mijenja smjer brzine, a iz grafa vidimo da tijelo nakon izvedenog manevra ima veću kinetičku energiju nego prije.

3.4 Primjeri uporabe u nastavi.

Klasična nastava fizike u kojoj učenik ima ulogu pasivnog primaoca informacija, čak i u slučaju najtalentiranijih predavača, rezultira niskim konceptualnim razumijevanjem fizike. Istraživanja su pokazala da u interaktivnoj nastavi u kojoj učenik ima aktivniju ulogu učenici postižu bolje rezultate [1]. Zbog toga se sve više koristi istraživački usmjerena nastava fizike. U istraživački usmjerenoj nastavi fizike učenik aktivno sudjeluje u nastavi te vođen nastavnikom dolazi do zaključaka, u usporedbi s tradicionalnim pristupom gdje samo treba reproducirati sadržaj. Istraživački usmjerena nastava Fizike započinje otvaranjem problema pokusom ili pitanjima koja novu pojavu ili koncept smještaju u kontekst realnog života. Nakon početne faze prikupljanja i diskutiranja ideja učenika slijedi važan korak upoznavanja nove pojave kroz pokus. Potom se postavlja jedno ili više istraživačkih pitanja na koja učenici nastoje odgovoriti kroz vođeno istraživanje, tj. planiranjem i provođenjem novih pokusa uz vodstvo nastavnika. Istraživanje najčešće ima za cilj otkriti pravilnosti karakteristične za pojavu koja se proučava i izgraditi model koji je može opisati, a može se provoditi kroz učeničke pokuse ili kroz frontalne istraživačke

pokuse i raspravu. U oba je slučaja izrazito važno da učenici postavljaju i testiraju hipoteze, predviđaju, grade modele, provode kontrolu varijabla, samostalno opisuju, organiziraju i ususatvljuju opažanja i rezultate mjerenja te ih predstavljaju ostatku razreda. U razrednoj se raspravi tada formiraju zaključci iz dobivenih rezultata. Uz nastavnikovu pomoć najčešće se tada formulira i matematički model koji opisuje novu pojavu, a potom se razmatra njegovo značenje i mogućnosti primjene. Sve navedene aktivnosti i procesi imaju veliku vrijednost i ulogu u razvijanju intelektualnih sposobnosti učenika, njihova prelaženja s konkretnog na formalno mišljenje, kao i za upoznavanje prirode znanosti [2]. Fizika proučava razne pojave, od kojih neke nije praktično proučavati na nastavi. U te pojave ubrajamo gibanje planeta čije proučavanje zahtjeva dugotrajna mjerenja, te kompliciranu opremu. U slučaju takvih tema odličan alat koji nam može pomoći u provođenju istraživački usmjerene nastave fizike su simulacije koje vjerno prikazuju neke prirodne pojave.

Simulacije Keplerovih zakona daju nam alat kojim učenici sami ili u grupama mogu izvoditi učeničke pokuse te sami doći do zaključaka, možda i neke formulacije Keplerovih zakona. Uz svaku simulaciju učenicima možemo postaviti nekoliko zadataka i istraživačkih pitanja ili im dati radni listić koji trebaju riješiti uz pomoć simulacije, ili ovo može biti istraživački projekt koji mogu raditi izvan nastave. Svaka od simulacija može poslužiti kao jedan pokus u kojemu se istražuje pojava, tako da ovaj program može poslužiti za 4 pokusa od kojih sa svakim možemo postaviti nekoliko istraživačkih pitanja:

Simulacija 1. Keplerovog zakona.

Pokrenite simulaciju te opišite što se događa?
Pokrenite simulaciju za različite parametre te proučite kako se putanja mijenja.
Kakvog su oblika putanje planeta oko Sunca?
Gdje se nalazi Sunce?
Ako su planet i Sunce približnih masa, što se događa sa Suncem?

Nakon izvođenja simulacija učenici bi uz pomoć nastavnika trebali primijetiti da se planeti oko Sunca gibaju eliptičnim putanjama, i da oblik ovisi o početnim uvjetima. Također mogu primijetiti da su moguće i kružne putanje. Dolaze do zaključka da se Suce nalazi u jednom od žarišta elipse. Ako je masa tijela koje kruži veće primjećuju da ono utječe na drugo tijelo te i ono počinje kružiti te dobivamo sustav poput binarnih zvijezda i

više nije jedno tijelo u žarištu nego je centar mase sustava ta dva tijela u žarištu elipse. Sve se uočeno onda može povezati s jednadžbom elipse

Simulacija 2. Keplerovog zakona.

Pokrenite simulaciju te opišite što se događa?
Kako brzina planeta ovisno o udaljenosti od sunca?
Kakve su iscrtane površine u odnosu jedne na druge?
Vrijede li prethodni odgovori i za neke druge putanje?

Nakon izvođenja simulacija učenici bi uz pomoć nastavnika trebali primijetiti da se planeti oko Sunca ne gibaju konstantnom brzinom, već kada je planet bliže Suncu on se brže giba.
Uočavaju da su prebrisane površine u jednakim vremenskim intervalima iste, što mogu vidjeti i iz grafa.

Simulacija 3. Keplerovog zakona.

Pokrenite simulaciju nekoliko puta opišite što se događa
Funkcija kojeg oblika bi dobro opisala točke na grafu?
Uočite kakav je odnos a^3/T^2?

Nakon izvođenja simulacija učenici bi uz pomoć nastavnika trebali primijetiti odnos ophodnih vremena i udaljenosti planeta oko Sunca. Točnije, trebali bi primijetiti da je taj odnos konstantan, vidjeti da kako raste udaljenost planeta od Sunca tako raste i njegovo ophodno vrijeme te da je taj odnos konstanta, što možemo vidjeti i iz grafa razdoblja T u odnosu na veliku poluos u kojem bi ravna linija prolazila kroz sve iscrtane točke pokazujući da je a^3/T^2 konstantan.
Simulacija gravitacijske praćke.

Pokrenite simulaciju nekoliko puta. Opišite što se događa?
Kakva je kinetička energija prije i poslije manevra?
Od kud je došla ili kuda je otišla ta energija?
Kako možemo dobiti različite putanje?

Nakon izvođenja simulacija učenici bi uz pomoć nastavnika trebali primijetiti da se nakon manevra poveća kinetička energija. Učenici dolaze do zaključka da dodatna energija dolazi od planeta oko kojeg se izvodi manevar te da zapravo i planet izgubi dio energije

no to je zanemarivo. Primjećuju da različitim ulaznim parametrima možemo utjecati na smjer i brzinu nakon manevra te da ovaj manevar može se može koristiti ne samo za dobivanje energije bez dodatne potrošnje goriva već i za mijenjanje smjera bez potrošnje goriva. A sve što je potrebno je dobro sve izračunati s kojim ulaznim parametrima trebamo ući u manevar da dobijemo željeni smjer i brzinu nakon.

Uz ovu simulaciju se mogu obrađivati i neke teme vezane za druge predmete kao što su matematika i informatika. Zbog jednostavnosti simulacije i korištenja Pythona koji se koristi u nastavi informatike ova simulacija se može koristiti i u nastavi informatike. Jedan od zadataka može biti da učenici sami ili u grupama, vođeni nastavnikom naprave simulaciju planetarnog gibanja. Za izvedbu simulacije dovoljno je samo znanje Newtonovih zakona te se u nekoliko linija koda može napisati funkcionalni dio simulacije koji je zadužen za izračune pozicije tijela u svakom trenutku. Iz Newtonovog zakona gravitacije izračunamo silu kojom jedno tijelo djeluje na drugo, a brzine i putanje onda pomoću sile dobijemo iz Newtonovih zakona gibanja. Također uz manje modifikacije možemo napraviti i simulacije gibanja nabijenih čestica što isto može biti zanimljivi učenički projekt.

Osim za teme vezane uz nastavu informatike ove simulacije se mogu koristiti i za teme vezane uz nastavu matematike, neki od primjera primjene su demonstracija raznih svojstava elipse te definiranje elipsi i njezinih svojstava, te kako elipsa ovisi o parametrima, računanje površine isječka elipse, ili aproksimacija cijele površine elipse, računanje ili mjerenje poluosi.

4. Zaključak

Programski jezik Python je odličan izbor za razne projekte zbog svoje jednostavnosti, rasprostranjenosti i mnogobrojnih paketa s gotovim programskim rutinama. Paket VPython nam daje alat za brzu i jednostavnu izradu 3d prikaza i iscrtavanje grafova. Koristeći VPython lako možemo izraditi simulacije koje vjerno opisuju neki stvarni sustav i njegovo ponašanje. Kako istraživački usmjerena nastava fizike daje bolje rezultate od klasične nastave u kojoj učenik ima ulogu pasivnog primaoca informacija, cilj nam je u nastavi fizike što više koristiti interaktivne metode. Kada nismo u mogućnosti neke pojave promatrati i istražiti putem pokusa ili kada su nam potrebne vizualizacije koje je teško postići tijekom pokusa tada možemo koristiti simulacije. Jedan primjer su planetarna gibanja čije bi promatranje dugo trajalo i zahtijevalo mnogobrojna mjerenja dok simulacija koja ih vjerno prikazuje traje nekoliko minuta. Također kada koristimo simulaciju za prikaz Keplerovih zakona možemo vizualizirati i dijelove koji ne bi bili vidljivi okom iz samog promatranja, kao naprimjer u slučaju 2. Keplerovog zakona površine koje prebriše radijus vektor.

S obzirom na sve veću dostupnost računala u većini učionica u Hrvatskoj, sve veći interes učenika za računala i razvoj programskih jezika koji svojim alatima olakšavaju izradu raznih simulacija kroz koje učenici mogu doživjeti određene pojave, simulacije postaju važan dio istraživački usmjerene nastave fizike.

Dodaci

Dodatak A Programska izvedba simulacije

```
1.  from vpython import *
2.  from random import random
3.
4.  # Definiranje konstanti.
5.  R = 15e9
6.  Re = 150e9
7.  Ms = 4e30
8.  Me = 6e24
9.  G = 6.67e-11
10.
11. # Definiranje globalnih varijabli.
12. global run
13. run = 0
14. sim = " "
15. global Object_A, Object_B, object_A_pos, object_A_mass, object_B_mass,
    object_B_velocity, r0, object_B_pos
16. global shownGraph, graphData, graphData2
17.
18. Object_A = sphere(
19.     pos=vector(0, 0, 0),
20.     radius=R * 2,
21.     color=color.yellow,
22.     make_trail=True,
23.     interval=50,
24.     retain=100,
25. )
26. Object_B = sphere(
27.     pos=vector(0, Re, 0),
28.     radius=R * 0.7,
29.     color=color.blue,
30.     make_trail=True,
31.     trail_type="points",
32.     interval=50,
33.     retain=100,
34. )
35.

36. def select_simulation(m):
37.     global sim
38.     sim = m.selected
39.     clear()
40.     InitGraphs()
41.     init()
42.
```

```
43. def Start(r):
44.     global run
45.     run = True
46.
47. def Stop(r):
48.     global run
49.     run = False
50.
51. # Inicijalizacija ispisa grafova ovisno o odabranoj simulaciji.
52. def InitGraphs():
53.
54.     global shownGraph, graphData, graphData2
55.
56.     if sim == "1. Keplerov zakon":
57.         shownGraph = graph(
58.             ymin=-10e11,
59.             ymax=10e11,
60.             xmin=-10e11,
61.             xmax=10e11,
62.             title=" ",
63.             xtitle="x",
64.             ytitle="y",
65.             width=400,
66.             height=400,
67.             foreground=color.black,
68.             background=color.white,
69.         )
70.         graphData = gcurve(
71.             graph=shownGraph, color=vector(random(), random(),
    random())
72.         )
73.         graphData2 = gcurve(
74.             graph=shownGraph, color=vector(random(), random(),
    random())
75.         )
76.     if sim == "2. Keplerov zakon":
77.         shownGraph = graph(
78.             width=600,
79.             height=500,
80.             title=" ",
81.             ytitle="area",
82.             xtitle="Period No",
83.             foreground=color.black,
84.             background=color.white,
85.         )
86.         graphData = gvbars(graph=shownGraph, delta=0.4,
```

```
                                color=color.green, label="bars")
    if sim == "3. Keplerov zakon":
        shownGraph = graph(
            width=600,
            height=500,
            title=" ",
            ytitle="Period",
            xtitle="Semi-major axis",
            foreground=color.black,
            background=color.white,
        )
        graphData = gdots(graph=shownGraph, color=color.black)
    if sim == "Gravitacijske praćka":
        shownGraph = graph(
            width=600,
            height=500,
            title=" ",
            ytitle="E<sub>k</sub>",
            xtitle="t",
            foreground=color.black,
            background=color.white,
        )
        graphData = gcurve(graph=shownGraph, color=color.red)

# Inicijalizacija početnih parametara ovisno o odabranoj simulaciji.

def init():
    global Object_A, Object_B, object_A_pos, object_A_mass,
object_B_mass, object_B_velocity, r0, object_B_pos

    object_A_pos = vector(0, 0, 0)
    object_A_mass = Ms
    Object_A.mass = object_A_mass
    object_B_mass = Me
    object_B_velocity = vector(30e3, 0, 0) * 1.5
    object_B_p = object_B_velocity * object_B_mass
    Object_B.mass = object_B_mass
    Object_B.velocity = object_B_velocity
    Object_B.p = object_B_p

    if sim == "Gravitacijske praćka":
        object_B_pos = vector(-2 * Re, Re, 0)
        Object_B.pos = object_B_pos

        object_A_velocity = -object_B_velocity / 15
        Object_A.velocity = object_A_velocity
        Object_A.p = Object_A.velocity * Object_A.mass
        r0 = Object_A.pos-Object_B.pos
    else:
```

```
134.         object_B_pos = vector(0, Re, 0)
135.         Object_B.pos = object_B_pos
136.         Object_A.p = -(Object_B.p)
137.
138. init()

139.
140. # Funkcija za brisanje ispisa.

141. def clear():
142.     global shownGraph
143.     if 'shownGraph' in globals():
144.         shownGraph.delete()
145.
146.     global object_A_oldpos, area, step, firstsector
147.     object_A_oldpos = vector(Object_B.pos)
148.     area = 0
149.     step = 0
150.     firstsector = 1
151.     Object_B.clear_trail()
152.     Object_A.clear_trail()
153.     chang_A_mass_slider.value = 0
154.     chang_B_mass_slider.value = 0
155.     chang_B_pos_x_slider.value = 0
156.     chang_B_v_x_slider.value = 0
157.     for obj in scene.objects:
158.         if obj is Object_A or obj is Object_B:
159.             continue
160.         obj.visible = 0
161.         del obj
162.
163. def Reset(r):

164.     clear()
165.     InitGraphs()
166.     init()
167.
168. # Funkcije za promjenu parametara.

169. def chang_A_mass(m):
170.     Object_A.mass = object_A_mass * (1 + m.value * 0.01)
171.     A_mass_text.text = "m<sub>A</sub> = " + \
172.         "{0:.1e}".format(Object_A.mass) + "\n "
173.     Object_A.clear_trail()
174.
175. def chang_B_mass(m):

176.     Object_B.mass = object_B_mass * (1 + m.value * 100)
177.     B_mass_text.text = "m<sub>B</sub> = " + \
```

```
178.         "{0:.1e}".format(Object_B.mass) + "\n "
179.     Object_B.p = Object_B.velocity * Object_B.mass
180.     Object_A.p = -(Object_B.p)
181.     Object_B.clear_trail()
182.
183. def chang_B_pos_y(r):

184.     Object_B.pos.y = object_B_pos.y * (1 + r.value * 0.1)
185.     B_distance_text.text = "r<sub>x</sub> = " + \
186.         "{0:.1e}".format(Object_B.pos.y) + "\n "
187.     Object_B.clear_trail()
188.
189. def chang_B_v(v):

190.     Object_B.velocity.x = object_B_velocity.x * (1 + v.value * 0.1)
191.     Object_B.p = Object_B.velocity * object_B_mass
192.     B_velocity_text.text = (
193.         "v<sub>x</sub> = " + "{0:.1e}".format(Object_B.velocity.x) +
    "\n "
194.     )
195.     Object_A.p = -(Object_B.p)
196.
197. # Postavljanje kontroli korisničkog sučelja.

198. chang_A_mass_slider = slider(
199.     pos=scene.title_anchor,
200.     top=10,
201.     value=0,
202.     length=300,
203.     min=-90,
204.     step=1,
205.     max=100,
206.     bind=chang_A_mass,
207. )
208.
209. A_mass_text = wtext(
210.     pos=scene.title_anchor,
211.     text="m<sub>A</sub> = " + "{0:.1e}".format(Object_A.mass) + " kg \n
    ",
212. )
213.
214. chang_B_mass_slider = slider(
215.     top=10,
216.     pos=scene.title_anchor,
217.     value=0,
218.     length=300,
219.     min=0,
220.     step=1,
221.     max=1000,
```

```
222.     bind=chang_B_mass,
223. )
224. B_mass_text = wtext(
225.     pos=scene.title_anchor,
226.     text="m<sub>B</sub> = " + "{0:.1e}".format(Object_B.mass) + " kg \n
     ",
227. )
228. 
229. chang_B_pos_x_slider = slider(
230.     top=10,
231.     pos=scene.title_anchor,
232.     value=0,
233.     length=300,
234.     min=-9,
235.     step=1,
236.     max=5,
237.     bind=chang_B_pos_y,
238. )
239. B_distance_text = wtext(
240.     pos=scene.title_anchor,
241.     text="r<sub>y</sub> = " + "{0:.1e}".format(Object_B.pos.y) + " m \n
     ",
242. )
243. 
244. chang_B_v_x_slider = slider(
245.     top=10,
246.     pos=scene.title_anchor,
247.     value=0,
248.     length=300,
249.     min=-9,
250.     step=1,
251.     max=10,
252.     bind=chang_B_v,
253. )
254. B_velocity_text = wtext(
255.     pos=scene.title_anchor,
256.     text="v<sub>x</sub> = "
257.     + "{0:.1e}".format(Object_B.velocity.x)
258.     + " m/s<sup>2</sup>\n ",
259. )
260. 
261. scene.append_to_title("\n Odaberite i pritisnite start :\n")
262. 
263. menu(
264.     choices=[
265. 
266.         "Odaberite simulaciju",
267.         "1. Keplerov zakon",
268.         "2. Keplerov zakon",
```

```
        "3. Keplerov zakon",
        "Gravitacijske praćka",
    ],
    index=0,
    pos=scene.title_anchor,
    bind=select_simulation,
)
cbutton = button(text="Zaustavi", pos=scene.title_anchor, bind=Stop)
cbutton = button(text="Pokreni", pos=scene.title_anchor, bind=Start)
cbutton = button(text="Resetiraj", pos=scene.title_anchor, bind=Reset)

# Postavljanje početnih vrijednosti varijabli

scl = 5
dt = 1000 * scl
counts = 0
t = 0
area = 0
object_A_oldpos = vector(Object_A.pos)
object_B_oldpos = vector(Object_B.pos)
number = 0
firstsector = 1
ccolor = vector(random(), random(), random())

# Simulacijska petlja
while 1:
    rate(1000)
    t = 0

    while run:
        t = t + dt
        r = Object_A.pos - Object_B.pos

        # U slučaju gravitacijske praćke se zaustavlja kada je izlazna udaljenost veća od ulazne,
        # a u slučaju Keplerovih zakona nakon jednog perioda.
        if sim == "Gravitacijske praćka":
            if mag(Object_A.pos - Object_B.pos) > mag(r0):
                run = 0
        else:
            if object_B_oldpos.x < 0 and Object_B.pos.x > 0:
                run = 0
                # Ako je 3. Keplerov zakon izračuna a^3/t^2
                if sim == "3. Keplerov zakon":
                    major_axis = major_axis_part_a+major_axis_part_b
                    semi_major_axis = major_axis/2
                    const = (semi_major_axis)**3 / (t**2)

                    time = float("{0:.1e}".format((t**2)*10e0))
```

```
316.                     sma = float("{0:.1e}".format((semi_major_axis**3) /
    Re))
317.
318.                     graphData.plot(sma, time)
319.
320.                     labelText = "  a<sup>3</sup>/T<sup>2</sup>=" + \
321.                         "{0:.1e}".format(const)
322.                     label(pos=Object_B.pos, text=labelText,
    color=vector(random(), random(), random()),
323.                           xoffset=10, yoffset=10)
324.                     major_axis_part_a = 0
325.                     major_axis_part_b = mag(Object_A.pos -
    Object_B.pos)
326.         rate((10**4) / scl)
327.         object_B_oldpos = vector(Object_B.pos)
328.         counts = counts + 1
329.         # Ako je 1. Keplerov zakon iscrta orbite tijela.
330.         if sim == "1. Keplerov zakon":
331.             graphData.plot(pos=(Object_A.pos.x, Object_A.pos.y))
332.             graphData2.plot(pos=(Object_B.pos.x, Object_B.pos.y))
333.
334.         if sim == "2. Keplerov zakon":
335.
336.             # Svaki  200-ti poziva funkcije ispiše radijus vektor za
    vizualizaciju površine.
337.             if not (counts % 200):
338.                 step = 0
339.                 curve(pos=[Object_A.pos, Object_B.pos],
    color=color.white)
340.                 ccolor = vector(random(), random(), random())
341.                 if not firstsector:
342.                     number = number + 1
343.                     label(
344.                         pos=Object_B.pos,
345.                         text="P= " + "{0:.1e}".format(area),
346.                         color=ccolor,
347.                         xoffset=10,
348.                         yoffset=10,
349.                     )
350.
351.                     graphData.plot(number, area)
352.                 firstsector = 0
353.                 area = 0
354.
355.             # Svaki  100-ti poziva funkcije računa površinu trokuta
    koji zatvori radijus vektor
356.             # površinama trokuta se aproksimira površina isječka.
357.             if not (counts % 100):
358.                 curve(pos=[Object_A.pos, Object_B.pos],
```

```
                    color=ccolor, emisive=True)
                x1 = Object_A.pos.x
                y1 = Object_A.pos.y
                x2 = Object_B.pos.x
                y2 = Object_B.pos.y
                x3 = object_A_oldpos.x
                y3 = object_A_oldpos.y
                att = 1 / 2 * abs(x1 * (y2 - y3) + x2 *
                                  (y3 - y1) + x3 * (y1 - y2))
                object_A_oldpos = vector(Object_A.pos)
                object_B_oldpos = vector(Object_B.pos)
                area = area + att
        # Pronalaženje glavne osi elipse.
        if sim == "3. Keplerov zakon":
            if mag(r) > major_axis_part_a:
                major_axis_part_a = mag(r)
            if mag(r) < major_axis_part_b:
                major_axis_part_b = mag(r)

        F = G * Object_A.mass * Object_B.mass * r.hat / mag2(r)
        Object_A.p = Object_A.p - F * dt
        Object_B.p = Object_B.p + F * dt
        ek = 1 / 2 * (Object_B.mass * mag2(Object_B.velocity))
        Object_A.pos = Object_A.pos + (Object_A.p / Object_A.mass) * dt
        Object_B.pos = Object_B.pos + (Object_B.p / Object_B.mass) * dt
        Object_B.velocity = Object_B.p / Object_B.mass

        if sim == "Gravitacijske praćka":
            graphData.plot(t, ek)

```

Literatura

[1] Hake, R. R. (1998). Interactive-engagement versus traditional methods: A sixthousand- student survey of mechanics test data for introductory physics courses. Am. J. Phys. 66, 64 – 74.

[2] FIZ_kurikulum.pdf (skolazazivot.hr) https://skolazazivot.hr/wp-content/uploads/2020/06/FIZ_kurikulum.pdf

[3] Dreyer, J.L.E. History of planetary systems from Thales to Kepler, Cambridge University Press, 1906

[4] General Python FAQ — Python 3.10.4 documentation, https://docs.python.org/3/faq/general.html#what-is-python ,26.4.2022

[5] Foreword for "Programming Python" (1st ed.) | Python.org, .https://www.python.org/doc/essays/foreword, 26.4.2022.

[6] Comparing Python to Other Languages | Python.org, https://www.python.org/doc/essays/comparisons/, 26.4.2022.

[7] Design of CPython's Garbage Collector - Python Developer's Guide, https://devguide.python.org/garbage_collector/, 26.4.2022.

[8] cpython/HISTORY at main · python/cpython · GitHub, https://github.com/python/cpython/blob/main/Misc/HISTORY 26.4.2022

[9] Sunsetting Python 2 | Python.org, https://www.python.org/doc/sunset-python-2 , 26.4.2022.

[10] Commits · python/cpython · GitHub, https://github.com/python/cpython/commits/2.7, 26.4.2022.

[11] Python Releases for Windows | Python.org, https://www.python.org/downloads/windows , 26.4.2022.

[12] What's New In Python 3.0 — Python 3.10.4 documentation, https://docs.python.org/3/whatsnew/3.0.html

[13] 2. Lexical analysis — Python 3.10.4 documentation, https://docs.python.org/3/reference/lexical_analysis.html#keywords, 26.4.2022.

[14] Built-in Types — Python 3.10.4 documentation, https://docs.python.org/3/library/stdtypes.html , 26.4.2022.

[15] 8. Errors and Exceptions — Python 3.10.4 documentation, https://docs.python.org/3/tutorial/errors.html , 26.4.2022.

[16] 4. More Control Flow Tools — Python 3.10.4 documentation, https://docs.python.org/3/tutorial/controlflow.html , 26.4.2022.

[17] Applications for Python | Python.org, https://www.python.org/about/apps/ , 26.4.2022.

[18] PYPL PopularitY of Programming Language indeks, https://pypl.github.io/PYPL.html , 26.4.2022.

[19] PyPI · The Python Package Indeks, https://pypi.org/ , 26.4.2022.

[20] NumPy, https://numpy.org/ 26.4.2022

[21] SciPy, https://scipy.org/ , 26.4.2022

[22] SymPy, https://www.sympy.org/en/index.html, 26.4.2022

[23] pandas - Python Data Analysis Library (pydata.org), https://pandas.pydata.org/ , 26.4.2022.

[24] Matplotlib — Visualization with Python, https://matplotlib.org/ , 26.4.2022.

[25] Project Jupyter | Home, https://jupyter.org/, 26.4.2022.

[26] Ministarstvo znanosti i obrazovanja - Okvirni godišnji izvedbeni kurikulumi za Nastavnu godinu 2020./2021. (gov.hr), https://mzo.gov.hr/vijesti/okvirni-godisnji-izvedbeni-kurikulumi-za-nastavnugodinu-2020-2021/3929 , 26.4.2022.

[27] VPython, https://vpython.org/ 26.4.2022.

[28] canvas (glowscript.org), https://www.glowscript.org/docs/VPythonDocs/canvas.html 26.4.2022

[29] graph (glowscript.org), https://www.glowscript.org/docs/VPythonDocs/graph.html 26.4.2022

[30] rate (glowscript.org), https://www.glowscript.org/docs/VPythonDocs/rate.html 26.4.2022

[31] Markley F. L., Crassidis J. L. Fundamentals of Spacecraft Attitude Determination and Control, Springer, 2014

[32] Brković N. Zbirka zadataka iz fizike, LUK d.o.o., 2001

[33] Morin, D. Introduction to Classical Mechanics: With Problems and Solutions. 1st ed. : Cambridge University press, 2008.

[34] Basics of Space Flight - Solar System Exploration: NASA Science https://solarsystem.nasa.gov/basics/chapter4-1/ 20.5.2022