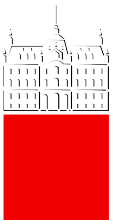
Univerza *v Ljubljani*
Fakulteta za *matematiko in fiziko*



Naloga 7: Newtonov Zakon

Matematično-fizikalni praktikum

Dimitrije Pešić

Vpisna številka: 28201072

Predavatelj: prof. dr. Borut Paul Kerševan

Ljubljana, 29.11.2023

1 Uvod

Naloga je od nas zahtevala, da preizkusimo uporabiti čim več metod za izračun nihanja matematičnega nihala z začetnim pogojem $\vartheta(0) = \vartheta_0 = 1, \dot{\vartheta}(0) = 0$. Poiskati smo morali korak, ki zadošča za natančnost na 3 mesta. Za razliko od prejšnje naloge smo se tu dodatno seznanili z tako imenovanimi *simplektičnimi* metodami.

2 Ozadje

Gibanje masne točke v polju sil v eni dimenziji opišemo z diferencialno enačbo drugega reda, z Newtonovim zakonom

$$m \frac{\mathrm{d}^2 x}{\mathrm{d}t^2} = F \, .$$

Enačba je seveda enakovredna sistemu enačb prvega reda

$$m \frac{\mathrm{d}x}{\mathrm{d}t} = p \, , \qquad \frac{\mathrm{d}p}{\mathrm{d}t} = F$$

in tako jo tudi rešujemo: kot sistem dveh enačb prvega reda.

Seveda morajo biti na voljo tudi ustrezni začetni pogoji, tipično $x(t = 0) = x_0$ in $dx/dt = v(t = 0) = v_0$. Splošnejše gre tu za sistem diferencialnih enačb drugega reda:

$$\frac{\mathrm{d}^n y}{\mathrm{d}x^n} = f(x, y, y', y'', ...),$$

ki ga lahko prevedemo na sistem enačb prvega reda z uvedbo novih spremenljivk v slogu gibalne količine pri Netwonovi enačbi ($y' = v, y'' = z, ...$).

Z nekaj truda se da eksplicitno dokazati, mi pa lahko privzamemo, da so metode za reševanje enačb hoda (Runge-Kutta 4. reda, prediktor-korektor...) neposredno uporabne za reševanje takšnih sistemov enačb in torej aplikabilne v poljubno dimenzijah, kar naj bi v principu zadovoljilo večino naših zahtev.

Obstaja še posebna kategorija tako imenovanih *simplektičnih* metod, za enačbe, kjer je f le funkcija koordinat, $f(y)$, ki (približno) ohranjajo tudi Hamiltonian, torej energijo sistema. Najbolj znana metoda je Verlet/Störmer/Encke metoda, ki je globalno natančna do drugega reda in ki točno ohranja tudi vrtilno količino sistema (če je ta v danem problemu smiselna). Rešujemo torej za vsak diskretni korak n velikosti h, $x_n = x_0 + n \cdot h$:

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = f(y)$$

in pri diskretizaciji dobimo recept za korak y_n in $v_n = y'_n$:

$$\begin{aligned} y_{n+1} &= y_n + h \cdot v_n + \frac{h^2}{2} \cdot f(y_n) \\ v_{n+1} &= v_n + \frac{h}{2} \cdot [f(y_n) + f(y_{n+1})] \, . \end{aligned}$$

Alternativno lahko to shemo zapišemo tudi s pomočjo dodatnih vmesnih točk in preskakujemo med lego in hitrostjo z zamikom h/2 (od tod angleško ime 'leapfrog' za ta zapis):

$$\begin{aligned} y_{n+1} &= y_n + h \cdot v_{n+1/2} \\ v_{n+3/2} &= v_{n+1/2} + h \cdot f(y_{n+1}). \end{aligned}$$

V še enem drugačnem zapisu je metoda poznana tudi kot metoda "Središčne razlike" (Central Difference Method, CDM), če nas hitrost ne zanima:

$$y_{n+1} - 2y_n + y_{n-1} = h^2 \cdot f(y_n),$$

kjer prvo točko y_1 izračunamo po originalni shemi. Metodo CDM lahko uporabljamo tudi za primere, ko je f tudi funkcija 'časa' x, f(x,y), le da tu simplektičnost ni zagotovljena (in tudi verjetno ne relevantna). Za simplektične metode višjih redov je na voljo na primer Forest-Ruth metoda ali Position Extended Forest-Ruth Like (PEFRL) metoda, ki sta obe globalno četrtega reda in enostavni za implementacijo.

3 Implementacija

Implementacije sem se zopet lotil v pythonu. Sprva sem poračunal za posamezne metode pri velikostjo koraka 10^{-2}. Na grafu 1 so prikazani rezultati integracije za Eulerjevo, Runge-Kutta-45 metodo, ki smo si ju pogledali že pri prejšnji nalgi, ter Verlet in Position Extended Forest-Ruth Like metodi, ki sta iz družine simplektičnih integratorjev.

Za primerjavo sem še dodal funkcijo *odeint* iz paketa *scipy.integrate*. Kasneje sem naletel na probleme pri uporabi *odeint* funkcije. Več o tem v naslednjem poglavju.

Vse metode sem primerjal z analitično rešitvijo matematičnega nihala:

$$x(t) = 2\arcsin\left[\sin\frac{x_0}{2}\mathrm{sn}\left\{K\left(\sin\frac{x_0}{2}\right) - \omega_0 t\ ;\ \sin^2\frac{x_0}{2}\right\}\right]\ , \tag{1}$$

kjer je $\omega_0^2 = \frac{g}{l}$ frekvenca, $g = 9.81\ m/s^2$ gravitacijski pospešek in l dolžina nihala. sn je Jacobijev eliptični integral in K popolni eliptični integral prve vrste. sn sem implementiral s funkcijo *scipy.special.ellipj*, K pa s *scipy.special.ellipk*.

Na sliki 1 lahko opazimo, da se rešitev Eulerjeve metode z časom kvari, kar je tudi lepo razvidno iz faznega diagrama. Namesto enotske elipse, se rezultati širijo po Arhimedovi spirali, kar nakazuje na slabo natančnost metode.

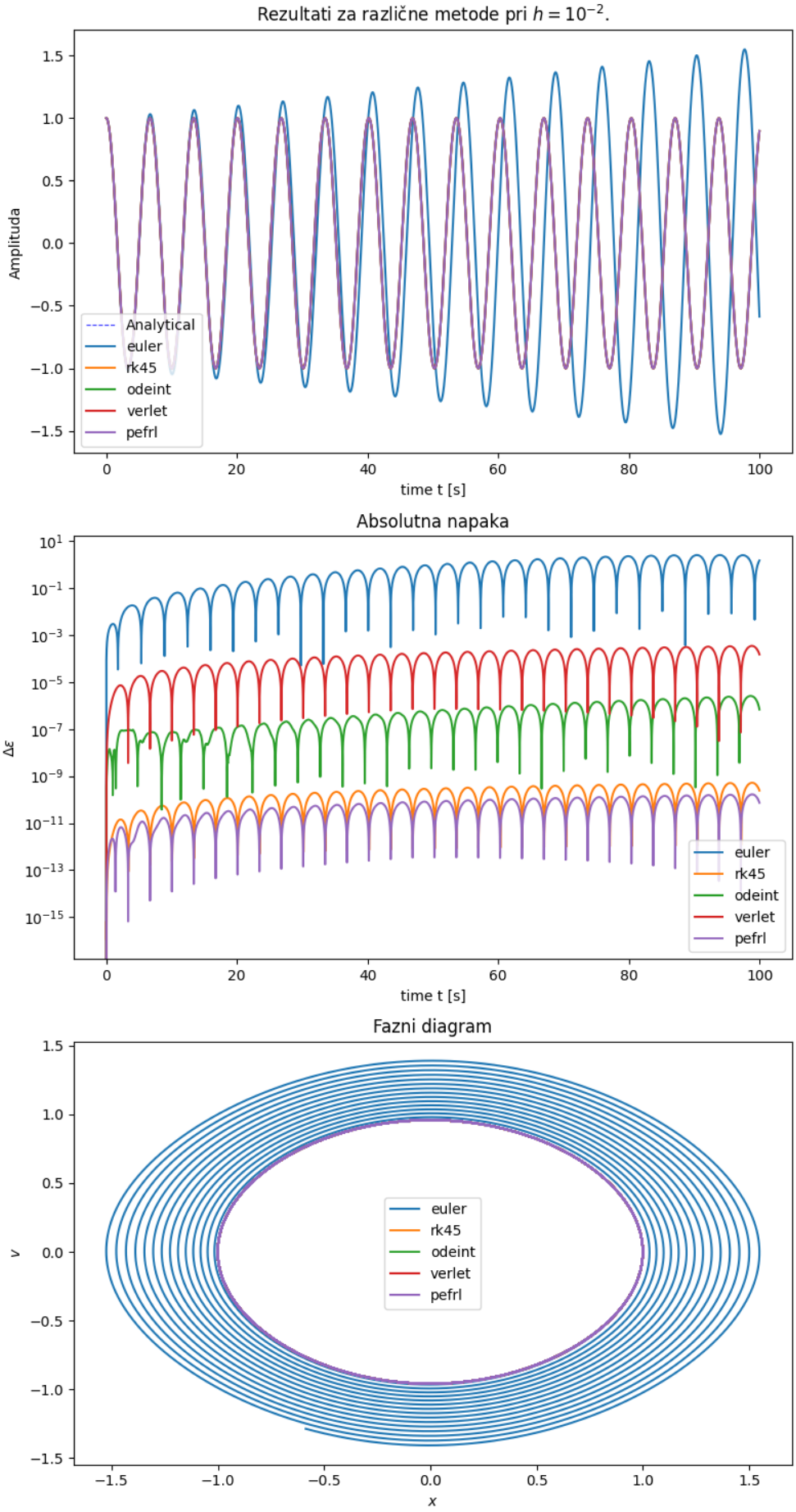
4 Energija

Za obravnavanje metod sem si še pogledal energijo, ki nam bolj nazorno pove, kako se sistem obnaša 2:

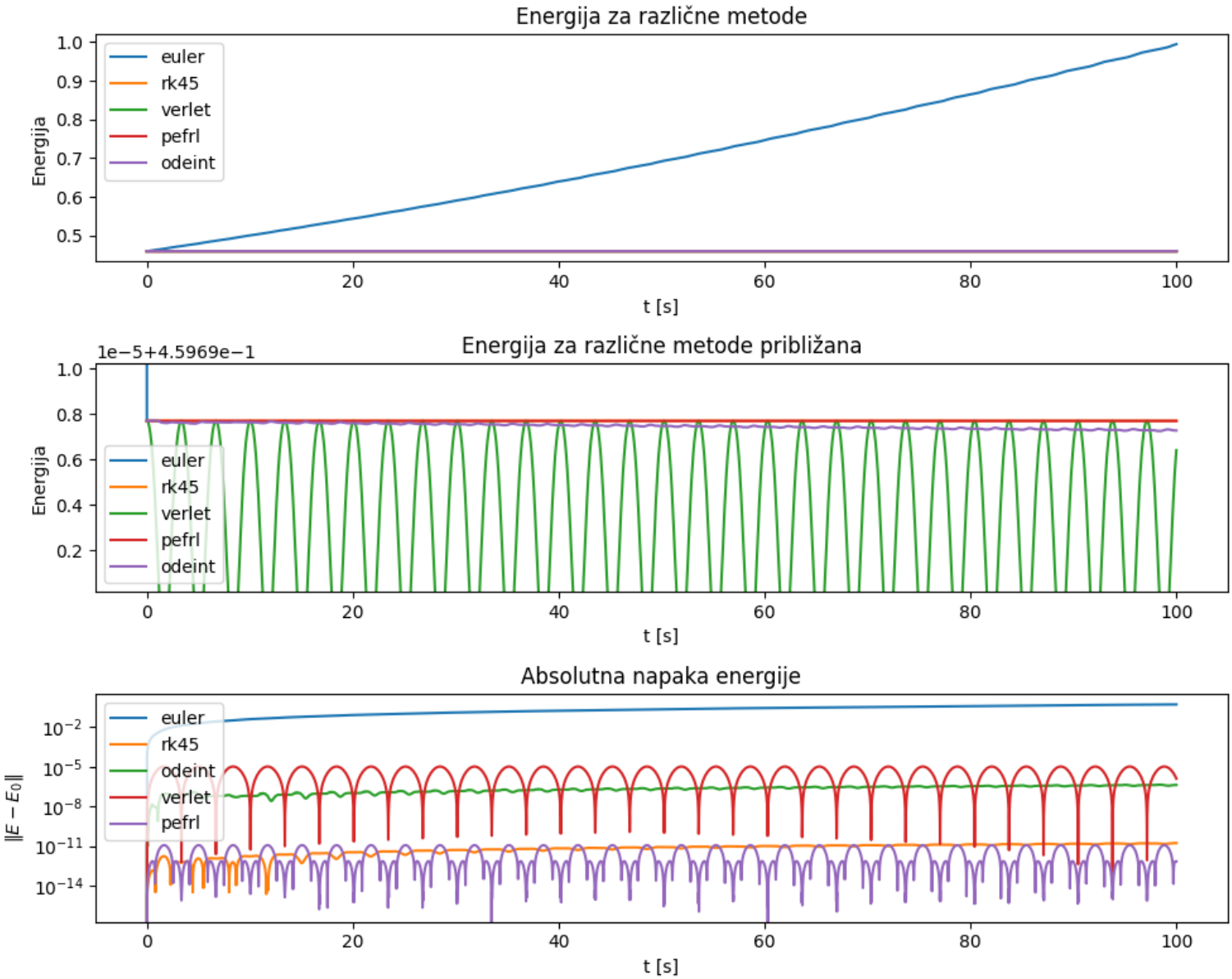
$$E \propto 1 - \cos(x) + \frac{\dot{x}^2}{2\omega_0^2}\ . \tag{2}$$

Pričakovali bi konstantno energijo, saj velja ohranitev energije pri matematičnem nihalu. Na grafu 2 lahko opazimo, poleg tega da je Eulerjeva metoda zelo nenatančna, da sta obe simplektčni metodi manj natančni kot Runge-Kutta-45 in Odeint (ki je pa le modificirana Runge-Kutta metoda).

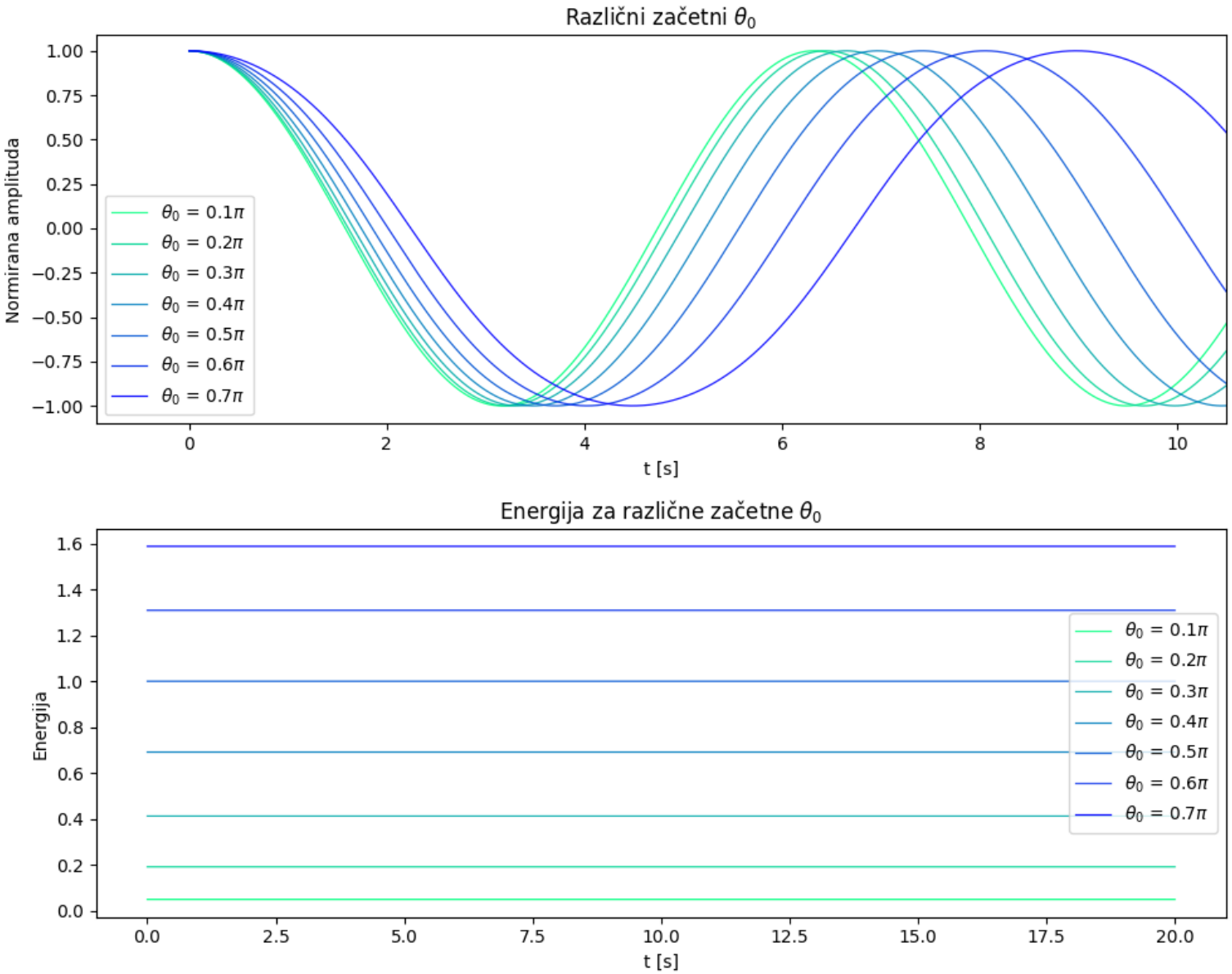
Za zabavo sem naredil sliko, ki prikaže normirane amplitude za različne začetne odmike 3.



Slika 1: Rezultati pri začetnem koraku $h = 10^{-2}$.



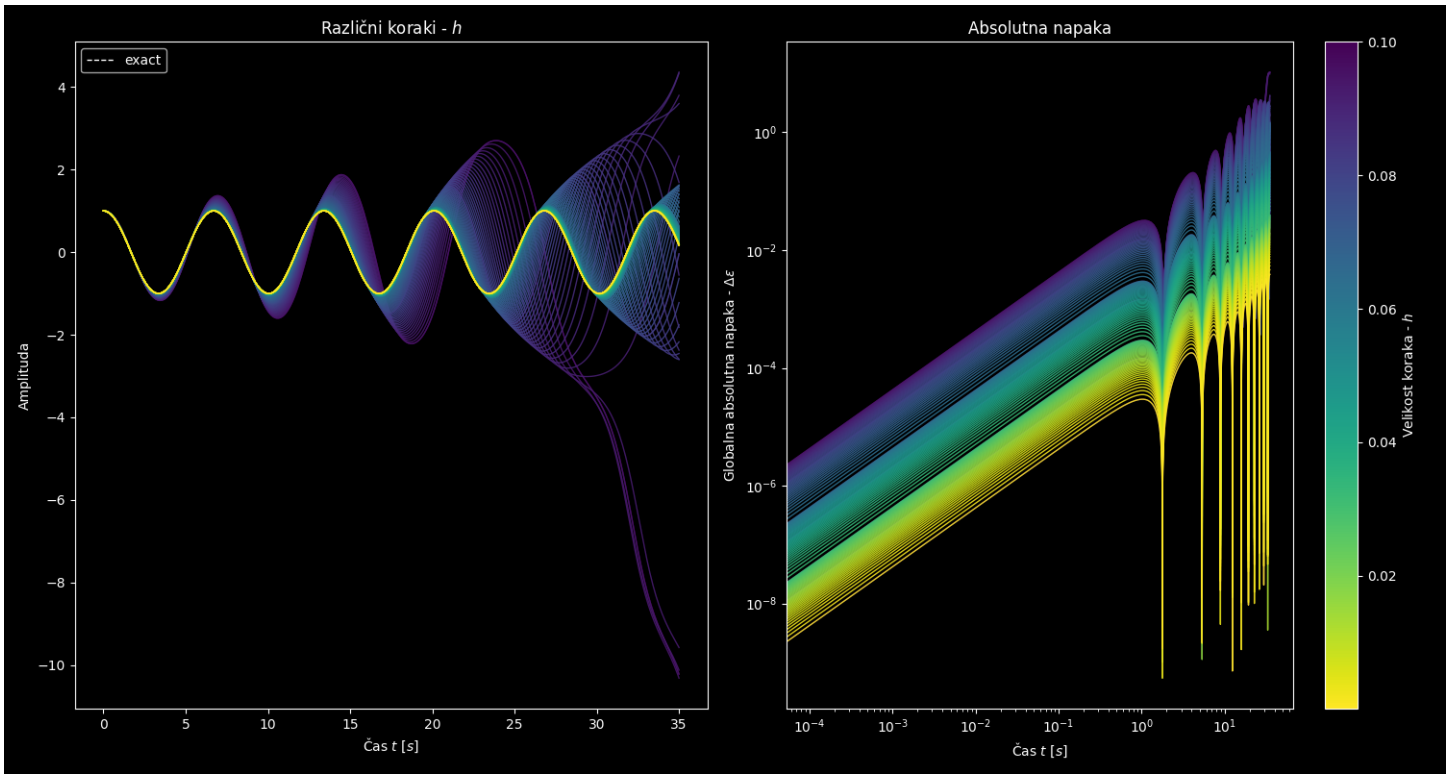
Slika 2: Izračunana energija za različne metode.



Slika 3: Rezultati za različne začetne odklone.

5 Natančnost in čas izvajanja

Kot pokazana v poglavju implementacija, se rešitve sistematično kvarijo s časom pri prevelikem koraku. Poglejmo si kako je natančnost Eulerjeve metode odvisna od velikosti koraka 4.



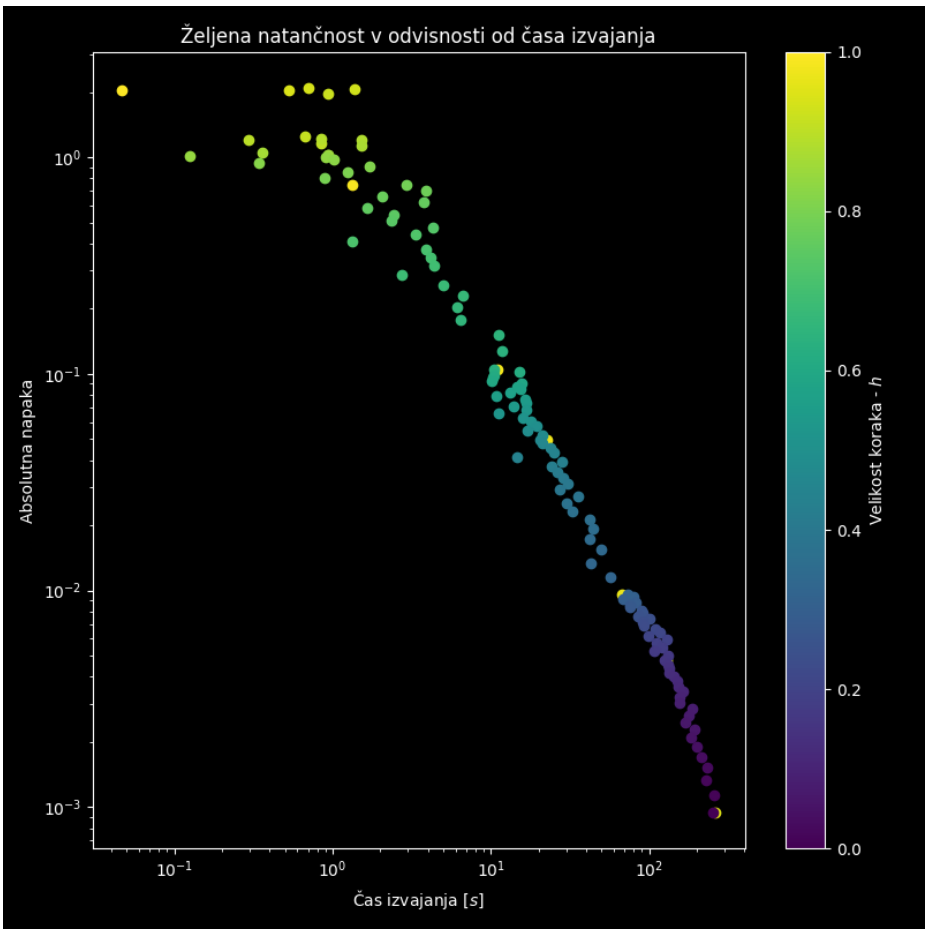
Slika 4: Kvarjenje rešitve pri Eulerjevi metodi.

Poglejmo si še čas izvajanja metod ter absolutna napaka od analitične rešitve, za različne vrednosti integracijskega koraka h. Prikazano na grafu 6.

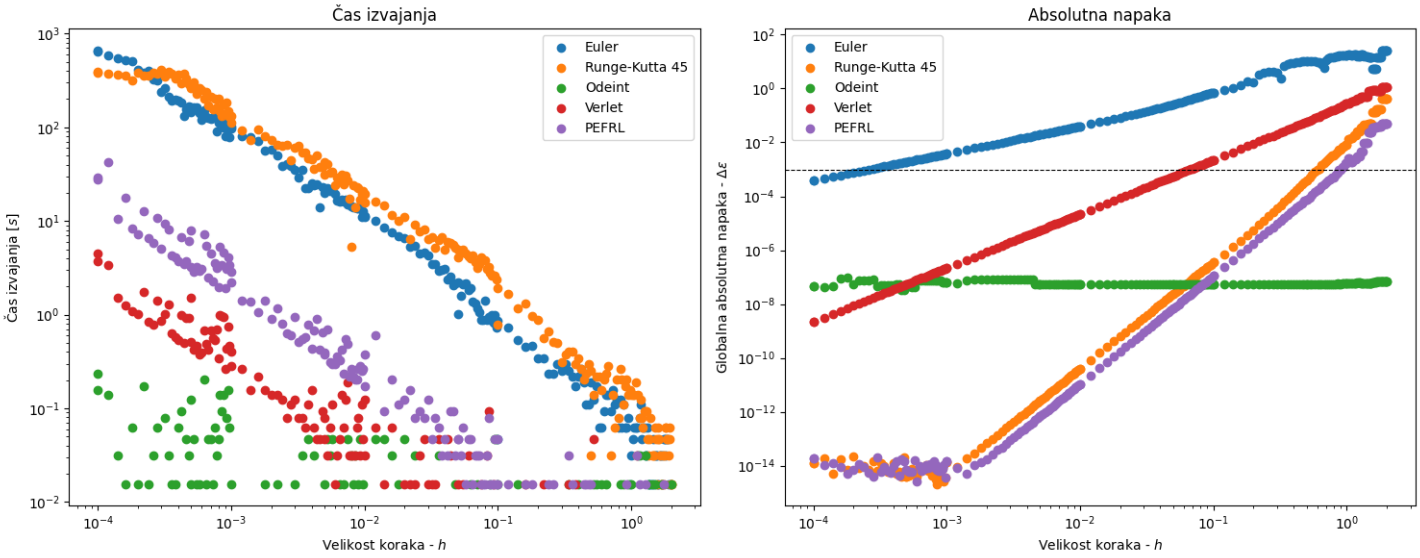
Iz grafa lahko razberemo potreben korak za željeno natančnost 10^{-3} 1. Opazimo lahko, da je RK45 še zmeraj kos bolj simplektični PEFRL metodi.

metoda	Euler	RK45	Verlet	PEFRL	Odeint
red potrebnega koraka	2.8×10^{-4}	3.4×10^{-1}	3.0×10^{-2}	6.5×10^{-1}	/

Tabela 1: Tabela reda velikosti koraka potrebnega za natančnost 10^{-3}.



Slika 5: Kvarjenje rešitve ter čas izvajanja.



Slika 6: Čas izračuna in natančnost v odvisnosti od koraka h.

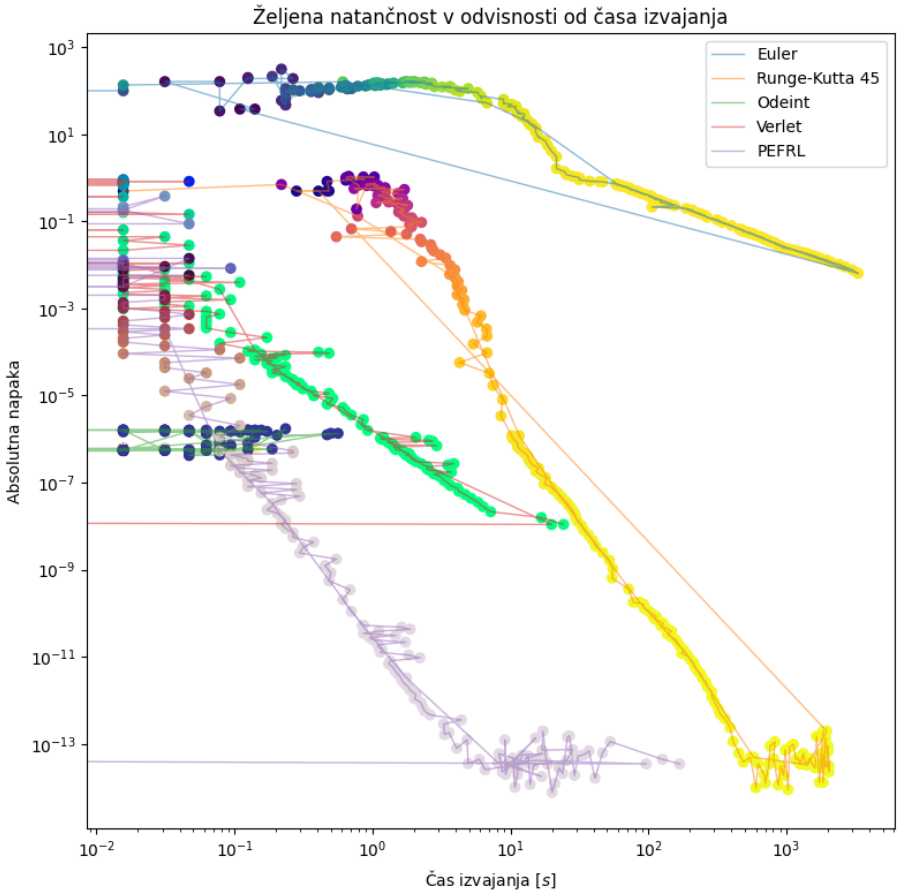
6 Bloopers

To poglavje sem namenil nekaj skoraj uspelim grafom tretje generacije, saj sem vložil veliko truda in časa v njih.

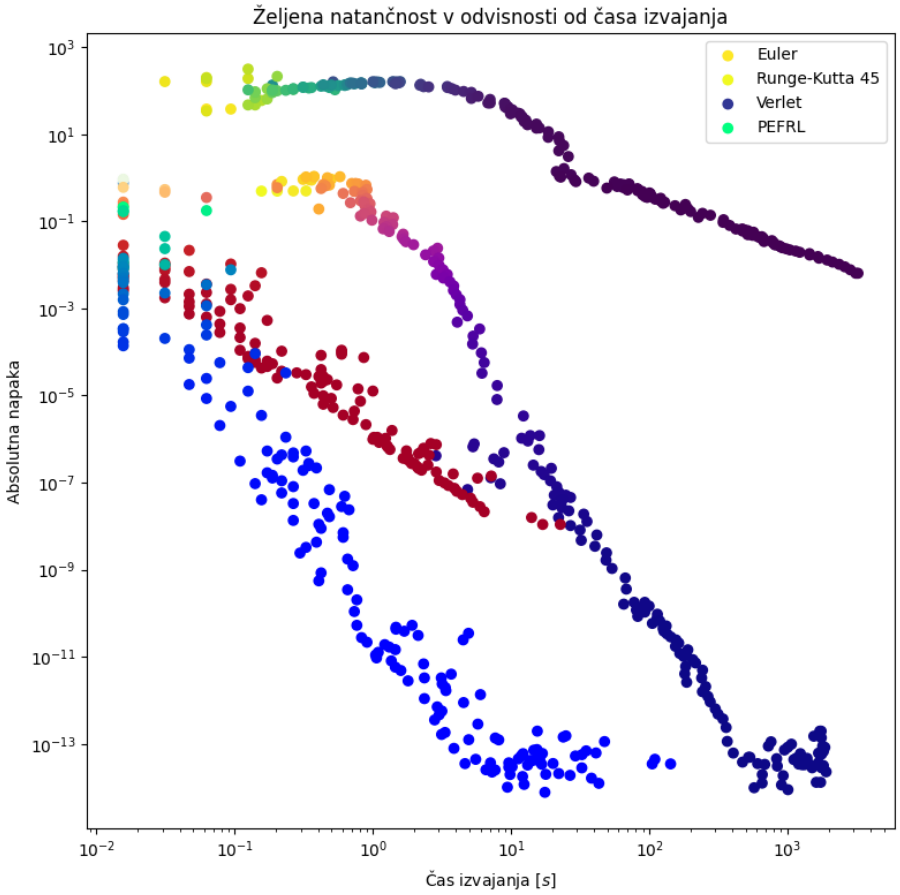
Sprva sem želel narediti kompakten graf, ki bi združil grafa na sliki 6, cilj je bil prikaz absolutne napake v odvisnosti od potrebnega časa izvajanja z barvnim gradientom, ki bi predstavljal velikost koraka. Poskusa sta prikazana na sliki 7 in 8. Pri grafu 7 nisem uspel pogruntati zakaj mi poveže črte tako čudno. Pri obeh me je mučila uporaba Odeint funkcije. Iz meni neznanega razloga (kljub precejšnjem času porabljenem na razhroščevanju) je ob večkratnem zagonu skripte, brez sprememb v le tej, klicalo ***SystemError*: null argument to internal routine**. Seveda, situacijo je poslabšala količina izračunov in čas potreben za le te. Zopet mi je na pomoč priskočila implementacija večnitnosti.

Malo mi je zmanjkalo časa, da bolj natančno obravnavam fazne diagrame, vendar sem vseeno nekaj izrisal. Na grafu je prikazano polje nevzbujenega matematičnega nihala 9

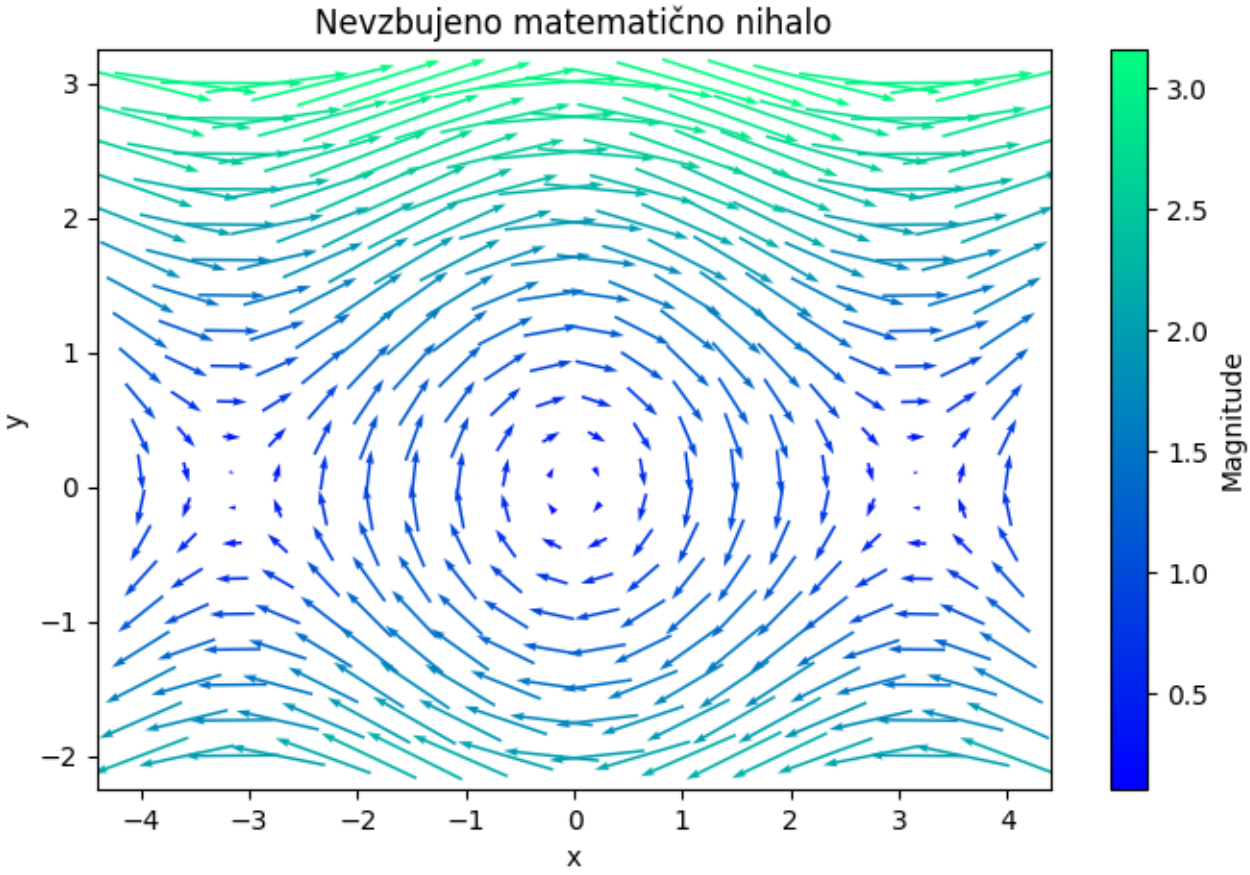
Dodatno se bom v nadaljnje bolj potrudil za bolje strukturiran compile v LaTeX-u.



Slika 7: Absolutna napaka v odvisnosti od časa izvajanja.



Slika 8: Absolutna napaka v odvisnosti od časa izvajanja.



Slika 9: Fazni portret matematičnega nihala.

7 Zaključek

Cilj naloge je bil, da se spoznamo z simplektičnimi integratorji in integriranje sistema linearnih diferencialnih enačb. Večino težav sem že komentiral. Tokratna naloga je napisana na malo krajše brez dodatkov. Želel sem obravnavati še dvojno nihalo in narediti kakšno lepo animacijo, alas, 'tis not meant to be. Morda pa se le pripravljam na stroge omejitve pri seminarju.